Intrusion Detection Using Payload Embeddings

MEHEDI HASSAN[1], MD ENAMUL HAQUE[2], MEHMET ENGIN TOZAL[1], VIJAY RAGHAVAN[1], RAJEEV AGRAWAL[3]
[1]School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70504, USA
[2]School of Medicine, Stanford University, Stanford, CA 94305, USA
[3]Information Technology Laboratory, U.S. Army Corps of Engineers, Vicksburg, MS 39180, USA

Corresponding author: Mehedi Hassan (e-mail: mehedi.hassan1@louisiana.edu).

ABSTRACT Attacks launched over the Internet often degrade or disrupt the quality of online services. Various Intrusion Detection Systems (IDSs), with or without prevention capabilities, have been proposed to defend networks or hosts against such attacks. While most of these IDSs extract features from the packet headers to detect any irregularities in the network traffic, some others use payloads alongside the headers. In this study, we propose a payload-based intrusion detection scheme, `PayloadEmbeddings`, using byte embeddings of the payloads of network packets. We employ a shallow neural network to generate vector representations for bytes and their corresponding payloads. Our feature extraction technique is coupled with the k-Nearest Neighbours (kNN) algorithm for the classification of packets as intrusive or non-intrusive. In our experiments, we evaluated 34 publicly available datasets, and used ten distinct payload-based, labeled intrusion detection datasets to train and evaluate our approach. Our empirical results show that `PayloadEmbeddings` reaches between 75% and 99% accuracy across all datasets. Finally, we compare our approach to other state-of-the-art and traditional intrusion detection techniques. Our findings suggest that `PayloadEmbeddings` demonstrates significant advantages over the other techniques on most of the datasets.

INDEX TERMS Intrusion Detection, Payload Embeddings, Byte Embeddings.

I. INTRODUCTION

Intrusion Detection Systems (IDSs) are used as parts of comprehensive defense mechanisms to protect systems from network-based attacks. Depending on the deployment type, IDSs are categorized into two types: Network-based and Host-based IDSs [1]. The Network-based Intrusion Detection Systems (NIDSs) are deployed at the network level and they inspect the incoming or outgoing traffic for any suspicious, abnormal, or malicious activity. The Host-based Intrusion Detection Systems (HIDSs) are deployed at each machine in the network to prevent malicious attacks targeting the hosts. HIDSs also help to prevent attacks coming from the machines within the network. Based on the detection method, IDSs are categorized into two types: Signature-based and Anomaly-based IDSs [2]. The Signature-based Intrusion Detection Systems (SIDSs) protect the network from malicious activities by inspecting packets in a network and comparing them against a known database of attack packet features. SIDSs fail to identify zero-day attacks. The Anomaly-based Intrusion Detection Systems (AIDSs) monitor the network traffic and compare it with the expected behavior of the network to generate an alert or not. AIDSs provide better protection against zero-day attacks, though they are not immediately tolerant to new behaviors.

Most of the IDSs use packet headers to extract features and apply machine learning methods for feature engineering and classification tasks. Such IDSs can successfully detect header-based attacks, e.g., scanning attacks or probing attacks. Moreover, the attacks that generate high volumes of packets are easily detected by header-based IDSs. However, these IDSs do not inspect the payloads of network packets. Hence, they fail to detect payload-based attacks such as SQL injection, shell-code, and cross-site scripting [3]. Unlike the header-based attacks, attackers send the payload-based attack packets at lower rates, as they try to exploit a vulnerability instead of overwhelming the network.

Prior research has been done using different techniques to detect anomalies in a network payload. Many use n-grams based approaches, e.g., PAYL [4] and ANAGRAM [5], which measure the occurrence frequencies of 256 possible byte values. The frequency distributions of bytes are used to develop a normal payload profile to compare against

https://doi.org/10.1109/ACCESS.2021.3139835

the incoming payloads for detecting attacks in a network. McPAD [6] uses a 2_v-gram technique to extract features from payloads. OCPAD [7], RANGEGRAM [8], HMMPayl [9] also use n-grams to extract features. Williams et al. [10] extracted 22 traffic-related statistical features from payloads. The authors use correlation and consistency-based feature selection techniques to reduce the feature space. However, payload features in these approaches either do not reflect any contextual information for bytes or reflect limited byte relationships.

Recently, researchers are using deep learning-based approaches to classify network payloads. Multiple studies have employed Convolutional Neural Network (CNN), and Recurrent Neural Networks (RNN) or Long Short-Term Memory (LSTM) [11]–[13]. Deep neural networks provide automatic feature extraction without manual feature engineering. However, they require a substantial amount of data and time to build an effective model against network attacks [14]. Additionally, these proposed models must take a fixed size input which requires truncating or padding payloads to a certain length. As a result, any contextual or semantic information is lost.

Our main goal is to develop an IDS that can identify known anomalies using payloads in network packets. To that end, we propose, `PayloadEmbeddings`, a high-level feature extraction technique employing contextual byte information for intrusion detection. Our model uses the payloads of network packets to map bytes to dense vector representations or embeddings. The model learns byte embeddings from their surrounding (context) bytes. Two bytes having similar contexts will have closer vector representations, which leads to natural groupings among the bytes that have similar contexts. Moreover, one can effectively aggregate the byte embeddings to obtain dense vector representations for their payloads. Our model can be deployed at both network-level, and host-level.

Word2vec, which has many successful applications in Natural Language Processing (NLP), generates continuous vector representations of words in high-dimensional space [15], [16]. We exploit the same idea to generate vector representations of bytes in packet payloads. We employ a shallow neural network to generate dense vector representations of bytes. The objective of the neural network is to maximize the log-probability of the neighboring bytes of each byte in payloads. Next, we aggregate the vector representations of bytes to build vectors representing their corresponding payloads, *i.e., payload embeddings*. Lastly, we use the vector representations of the payloads as features feeding a k-Nearest Neighbours (kNN) classifier that labels target packets as intrusive (anomalous) or non-intrusive (normal). kNN classifiers are able to generate convoluted boundaries, especially when the data instances are dispersed and not easily separable.

What is more, a suitable benchmark dataset is required to train and evaluate intrusion detection systems [17]. Publicly available, labeled datasets with raw payloads are essential for payload-based intrusion detection systems, such as our model. The datasets with real network activities containing both attack and normal traffic are quite difficult and expensive to obtain [6]. In the last few decades, a large number of IDS studies have been done on DARPA 1998/99 [18], [19], and KDD-Cup 99 datasets. The DARPA 1998/99 datasets have been widely used in various payload-based intrusion detection systems [4]–[6], [9], [13]. However, these datasets have often been criticized for outdated DDoS attacks, artificial attack injections, and redundancy [20], [21]. Finding a suitable dataset with modern real-world attacks and normal traffic is a challenge in the arena of payload-based intrusion detection systems research. We performed an exhaustive search for datasets to evaluate our approach. We found ten labeled datasets with payloads that are suitable for this study, out of 34 datasets in total. Specifically, we evaluate our model on Botnet, CIC DoS, CICIDS-2017, CSIC HTTP 2010, CTU-13, ECML/PKDD 2007, ISCX-2012, ISOT, NDSec-1, and UNSW-NB15 datasets. The detailed descriptions of these datasets are given in Section IV.

Our empirical results indicate that `PayloadEmbeddings` reaches 92%-99% accuracy, precision, recall, and F1-score on ISOT, UNSW-NB15, CICIDS-2017, and CIC DoS datasets, and 75%-82% accuracy, precision, recall, and F1-score on the other datasets, using kNN. We compare our approach to ten other traditional and state-of-the-art techniques, including PAYL [4], McPAD [6], HMMPayl [9], AEIDS [12], HAST [11], OCPAD [7], EsPADA [22], CBID [23], PL-RNN [13], and Packet2Vec [24] over the same datasets and show that our approach performs better over most of the datasets.

Our main contributions in this study are listed as follows:

- We propose to employ a shallow neural network, *Word2Vec*, to generate vector representations of bytes, *i.e., byte embeddings*, from the payload of network packets. Next, we utilize the *byte embeddings* corpus model to generate payload vectors, *i.e.*, `PayloadEmbeddings` that can be used as features for the classification task.
- We evaluated `PayloadEmbeddings`, and ten other state-of-the-art and traditional feature extraction techniques on ten appropriate datasets out of 34 publicly available datasets. Additionally, we share the implementation of our proposed technique and the other ten methods online[1].

The rest of the paper is organized as follows. Section II presents the related work. Section III describes the proposed model, `PayloadEmbeddings`. Section IV presents an overview of the intrusion detection datasets used in this study. Section V presents experimental settings of `PayloadEmbeddings`. Section VI describes the analyses of the experimental results. Section VII discusses the limitations of `PayloadEmbeddings`. Finally, we conclude our work in Section VIII.

[1]BitBucket: Payload Embeddings

II. RELATED WORK

In this section, we categorize the previous studies based on their feature extraction techniques.

A. TRADITIONAL FEATURE EXTRACTION TECHNIQUES

N-gram and Frequency. Many payload-based intrusion detection systems employ n-gram based feature extraction techniques. *PAYL* [4] uses the byte frequency distributions of normal packets to develop a centroid model. Relative frequency is used as a feature vector applying the n-gram (n=1) approach. PAYL adopts the Mahalanobis Distance (MD) map to compute the consistency of incoming payloads. If the new payload exceeds a threshold, it is classified as anomalous. Wang et al. proposed ANAGRAM [5] to deal with polymorphic blending attacks. ANAGRAM develops a bloom filter from n-grams in normal payloads. N-grams are extracted from incoming payloads to identify the occurrence of new n-grams in the bloom filter. If the occurrence of such n-grams exceeds a certain percentage, the payload is labeled as anomalous. Vidal et al. proposed EsPADA [22] to protect networks against adversarial threats. The proposed approach employs the n-gram technique to extract features from the payloads. The authors utilized Counting Bloom Filters (CBF) to store the extracted features. OCPAD [7] also uses n-gram technique to extract sequences from payloads. The authors propose a knowledge-based data structure named *Probability Tree* to store the occurrence probability range of n-grams from normal payloads. RANGEGRAM [8] considers the maximum and minimum occurrence frequency of n-grams to detect zero-day attacks against web traffic. A database is generated from the n-grams of normal traces. If an incoming packet has a deviation from the database, it generates an alert. McPAD [6] uses a modified n-gram technique to extract the features from payloads. The authors have adopted the n_v-gram technique during the training phase where the occurrence frequency of each n number of bytes separated by length v, is calculated. McPAD generates different feature spaces by varying the value of v. In the testing phase, multiple one-class Support Vector Machine (SVM) is used to detect anomalous packets by majority voting. These approaches exploit character frequency distributions in payloads as features, which often do not reflect the context-level information.

N-gram and Neural Network. *Packet2Vec* [24] utilizes n-grams ($n = 2$) to extract sequences of bytes from packets. The authors also employ *Word2Vec* to create vector representations for each of the *most-frequent* n-grams. The vectors of n-grams for each packet are divided by the number of n-grams found in that packet to create a fixed-size packet vector.

Packet2Vec also exploits *Word2Vec* similar to the `PayloadEmbeddings`. However, there are some major differences. The vocabulary size of *Packet2Vec* is $2^{16} = 65,536$ and the vector length is 128. On the other hand, the vocabulary size and the vector length are only $2^8 = 256$, and 10 in `PayloadEmbeddings`, respectively. As a result, the time-complexity and memory consumption for *Packet2Vec* are way higher than `PayloadEmbeddings`.

B. DEEP NEURAL NETWORK BASED TECHNIQUES

HAST-IDS [11] uses CNN and LSTM to capture low-level spatial and high-level temporal features from packets, respectively. AEIDS [12] employs an Autoencoder to detect outliers in network traffic. A reconstruction error on normal traffic, and a modified z-score are used to classify the incoming traffic. Liu et al. [13] proposed two models for payload-based intrusion detection, *i.e.,* PL-CNN and PL-RNN. The authors utilized deep learning models to learn features from payload without manual feature engineering. However, the proposed approach is evaluated using the outdated DARPA 1998/99 dataset which has become obsolete in time. As these methods rely heavily on the deep learning models for feature extraction and classification tasks, the time complexity is higher than the other IDS techniques [13]. Compared to deep learning models, our proposed method employs a shallow neural network (with one hidden layer) only for byte embeddings generation. Hence, our method is computationally faster.

C. OTHER TECHNIQUES

PCNAD [25] considers the specific content of a payload. Content-based partitioning (CPP) is used to determine payload profiles for different lengths of payloads. Applying the CPP technique, PCNAD uses 62.64% of the full payload length, on average. Hosseini et al. [23] proposed a payload-based attribution scheme named CBID (Compressed Bitmap Index and Traffic Downsampling). CBID extracts features from down-sampled traffic using the combination of bloom filters and compressed bitmap index table. Jamdagni et al. [26] propose a 3-tier feature selection technique named Iterative Feature Selection Engine (IFSEng). In the first tier, Principal Component Analysis (PCA) is used to analyze raw data. Tier 2 computes the number of dominant Principal Components (PCs). Finally, a normal model is generated, and trained in tier 3. The authors use Mahalanobis Distance (MD) map to extract correlations between the packets and between the features. Like PAYL [4], (MD) map is used to classify payloads as anomalous or normal. Luo et al. [27] developed a combined model with XGBoost and PU learning for web anomaly detection. This method vectorizes HTTP payloads at the byte-level by ASCII values to avoid information loss. PU learning trains a binary classifier. Naive Bayes (NB) and Logistic Regression (LR) have been used to identify malicious behaviors. The combined model performs best when the vector dimension is 7500. However, due to multiple stages of payload analysis, the proposed approach is computationally costly and do not preserve contextual relationships between bytes in payloads.

Unlike the previous studies [4]–[8], [11]–[13], [22], [23], [23]–[27], `PayloadEmbeddings` can extract contextual information from payloads. Each byte in a payload is transformed into a vector space by maximizing the log probability

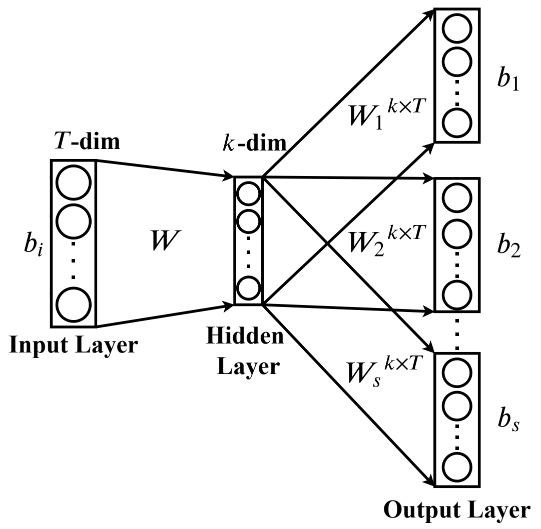


FIGURE 1: The shallow neural network architecture predicting context bytes given an input byte.

of the neighborhood bytes with a window size of 5 to capture the contextual relations among the adjacent bytes. Moreover, the vocabulary size (=256) limits the time complexity, making our approach computationally faster than others.

III. PAYLOAD EMBEDDINGS CORPUS MODEL

In this section, we first briefly present *word2vec* models that inspired us to develop byte and payload embeddings for intrusion detection. Then, we introduce our model, `PayloadEmbeddings`, in detail.

Continuous bag of words (*CBOW*) and *Skip-gram* models are together known as *word2vec* [15]. *Word2vec* is an efficient natural language (NLP) model to learn vector representations for words or embeddings. The most practical aspect of word embeddings is to create dense vector representations of words that contain semantic meanings. The *CBOW* model predicts a word given a context as input, while the *Skip-Gram* model predicts the context given a word as input. It is a self-supervised learning framework that learns distributed representations of texts of any length. The Paragraph vector model [28] is an extension of *word2vec* which is also known as *doc2vec*. The *doc2vec* model creates embeddings or vector representations for paragraphs. To obtain paragraph vectors, word embeddings are typically aggregated by concatenation or coordinate-wise averaging.

We extend the *Skip-gram* model to generate a corpus of embeddings for the bytes in packet payloads. We build byte embeddings corpus model from the output of the hidden layer of the model. Then, for each payload in our dataset, we generate a feature vector using the aggregated vectors of individual bytes from the byte embeddings corpus. Note that, we consider complete payloads instead of trimming or padding the payloads to a fixed length during the training of the corpus model.

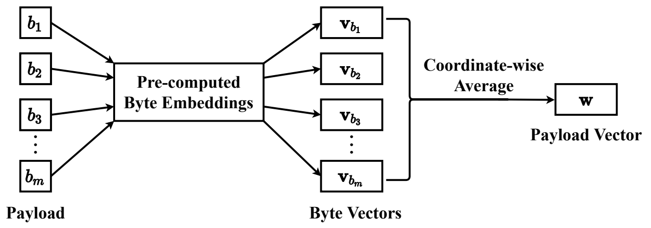


FIGURE 2: Average aggregation generating a payload embedding from pre-computed byte embeddings.

A. BYTE EMBEDDINGS GENERATION

Figure 1 demonstrates the shallow neural network architecture, similar to *Skip-gram* for NLP, to generate byte embeddings. The input layer takes a T-dimensional one-hot encoded input byte b_i where T=256 unique bytes in the corpus. The hidden layer consists of k neurons to produce k-dimensional vector representation of the input byte, b_i. The training objective of the model is to generate contextual byte representations for an input byte b_i with respect to its neighboring bytes within a predefined window size, s. Considering n training bytes $b_1, b_2, \ldots, b_n$, the objective is to maximize the average log-likelihood denoted as follows:

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{-s\leq j\leq s, j\neq 0}\log p(b_{i+j}|b_i), \tag{1}$$

where s is the window size of the input byte b_i. For example, if window size, $s = 3$, then for each byte b_i we compute the average of the log probability of b_{i-3} to b_{i+3}, except the input or center byte b_i. Averaging the log-likelihood would generate more stable byte representations. he probability of $p(b_{i+j}|b_i)$ in (1) is defined using the softmax function in (2):

$$p(b_c|b_i) = \frac{\exp\left(\mathbf{v}'_{b_c}{}^{\top}\mathbf{v}_{b_i}\right)}{\sum_{j=1}^{T}\exp\left(\mathbf{v}'_{b_j}{}^{\top}\mathbf{v}_{b_i}\right)}, \tag{2}$$

where b_c, $\mathbf{v}_b$, and $\mathbf{v}'_b$ are the surrounding bytes, input and output vector representations of byte b_i, respectively, and T is the total number of unique bytes in the payload corpus. Note that, the computation of gradient $\nabla\log p(b_{i+j}|b_i)$ of the objective function of (1) is expensive in the original *skip-gram* model. However, our payload dataset contains a maximum of 256 unique bytes which limits the computational complexity. By taking the output of the hidden layer, we generated a corpus model for bytes of all payloads in our dataset.

B. PAYLOAD EMBEDDINGS GENERATION

The low dimensional feature vector generation approach for payloads is depicted in Figure 2. The figure illustrates vector aggregation of a payload containing m bytes b_1, b_2, ... , b_m. For each byte b_i, the corresponding vector v_i of length k is selected from the pre-computed byte embeddings generated by the *skip-gram* model (Figure 1). The vectors are then averaged to get the final payload feature vector with dimension k. We use the coordinate-wise vector aggregation method [16] by using (3):

$$\mathbf{w}[i] = \frac{1}{m}\sum_{j=1}^{m} \mathbf{v}_{b_j}[i], \tag{3}$$

where, $\mathbf{v}_{b_j}[i]$ denotes the i^{th} element of the byte embedding of the j^{th} byte. m is the number of bytes in the payload and $\mathbf{w}[i]$ is the i^{th} element of the final payload feature vector, $\mathbf{w}$. Note that, vector length k refers to the feature size.

Finally, the payload vectors are used as features and fed into our classification algorithm, k-Nearest Neighbours (kNN).

Time Complexity. The time complexity for training the byte embeddings corpus model is $\mathcal{O}(s*(N+N*T))$, where s is the context window size, T is the number of unique bytes, and N is the total number of bytes in the training dataset [15], [29]. The time complexity to generate a feature vector, *i.e.,* payload embeddings is $\mathcal{O}(m)$, where m is the number of bytes in a payload in the testing dataset. Note that the byte embeddings are generated only once in the training stage and the feature vectors of payloads are computed from the pre-computed byte embeddings as packets arrive during the testing stage.

Scalability. In the training stage, we learn byte embeddings from the payloads of the packets in the training dataset. In Section VI-A, we show that the byte embeddings generation process takes 5,300 to 508,000 ms for different datasets. The byte embeddings model generation is executed offline and only once. Once we learn the byte embeddings, payload embedding vectors are computed online using (3) as new packets arrive. In Section VI-A, we show that the payload generation takes between 1.09ms to 3.46ms on average over different datasets. Unlike Intrusion Prevention Systems (IPSs), Intrusion Detection Systems (IDSs) are typically not deployed inline. They passively monitor the network traffic flow and report the anomalies in the form of notifications to the administrators. To handle ever-increasing network throughput, modern IDSs take advantage of parallel computing, multi-threading, and GPU-accelerated computing. `PayloadEmbeddings` can also process large amounts of payloads in short times as part of these modern IDSs. Hence, the proposed approach is scalable.

IV. DATASETS

Most of the IDS datasets contain only the headers of network packets. Many datasets using real traffic either do not have payloads or payloads have been removed due to privacy and security reasons [17]. Lack of labeled datasets with real network traffic is a significant issue in the area of payload-based anomaly detection research. As a result, anomaly detection using payloads has been evaluated on private datasets by most researchers. However, such datasets are difficult to collect and sometimes unavailable due to privacy concerns. Since `PayloadEmbeddings` is a self-supervised learning method, we need well-labeled datasets with raw payloads. In search of a suitable dataset, we came across several IDS datasets. Our search criteria for appropriate datasets were set to raw, anomalous, and labeled "payloads". Table 1 summarizes the features of the ten datasets that we use in this study. These features include a subset of the 15 important features to assess intrusion detection datasets introduced by Ring et al. [17]. The features consist of the year of creation, availability (whether the dataset is publicly available, restricted, or private), the format that the dataset is available in, the type of network traffic trace (real, emulated, or synthetic), size of the datasets, whether the dataset is labeled or not, whether the dataset contains metadata or not, payload availability, if the dataset is balanced or not, and the type of attacks. In the following, we briefly provide the descriptions of these datasets.

Botnet: Beigi et al. created the Botnet [30] dataset in 2014 using three existing datasets: ISOT [31], ISCX 2012 [32] and CTU-13 [33]. The dataset contains various attacks from botnets such as IRC bot, Virut, Black Hole, Neris, Menti Rbot, Tbot, Murlo, Sogou, Weasel, Nsis, Zeus, and Zero-access. Malicious IP addresses are provided as ground truth for labeling the packets. The dataset is provided in a packet-format. Payloads are present in the traffic capture. The dataset is publicly available in two parts; training and test sets. We use this dataset in our study because it contains labeled payloads.

CIC DoS: The Canadian Institute for Cybersecurity created CIC DoS [34] dataset in 2012. The dataset contains eight different HTTP-based application layer DoS attacks. The attack traffic was generated using Ddossim, Goldeneye and, Hulk. Normal traffic was generated from non-attack traffic of the ISCX 2012 dataset. The dataset is recorded in packet format, labeled and is publicly available. Since this dataset meets our dataset criteria of labeled packets with payloads, we use it in this study.

CICIDS-2017: Sharafaldin et al. published the CICIDS-2017 [35] dataset. This dataset is created over a duration of 5 days in 2017. The dataset is available in both packet and bi-directional flow-based format and contains seven common updated family of attacks. The authors have provided additional meta-data about IP addresses and attacks. The attacks present in this dataset are Brute Force, Heartbleed, Botnet, DoS, DDoS, Infiltration and Web Attack. This dataset is well labeled and publicly available. We use the CICIDS-2017 dataset in our study, as it meets our dataset criteria.

CSIC HTTP 2010: Information Security Institute of CSIC created the CSIC HTTP 2010 dataset [36]. This dataset contains web requests which are automatically generated. The dataset has three files: training with normal traffic, testing with normal traffic, and testing with anomalous traffic. The dataset only provides payloads, the packet headers are removed. Three types of anomalous requests are included in this dataset; Static attacks, Dynamic attacks and, Unintentional illegal requests. However, the dataset is only labeled as anomalous and normal. Payloads are not labeled as a certain

TABLE 1: A short overview of the intrusion detection datasets used in this study.

Dataset	Year of Creation	Availability	Format	Labeled	Traffic Type	Meta Data	Size Count	Payload	Balanced	Attack Type
Botnet	2010	Public	Packet	Yes	Emulated	Yes	14GB packets	Yes	No	Botnets (IRC bot, Black Hole, Neris, Virut, Menti, Rbot, Tbot, Murlo, Sogou, Weasel, Nsis, Zeus, and Zero-access)
CIC DoS	2012	Public	Packet	Yes	Emulated	No	4.6GB packets	Yes	No	Ddossim, Hulk, Slowhttptest, Rudy, Goldeneye, Slowread, Slowbody, and Slowloris
CICIDS 2017	2017	Public	Packet, Flow	Yes	Emulated	Yes	3.1M flows	Yes	No	Brute Force, Heartbleed, DoS, Botnet, DDoS, Infiltration, and Web Attack
CSIC HTTP 2010	2010	Public	Packet	Yes	Emulated	No	61k packets	Yes	No	Static attacks, Dynamic attacks, and illegal request
CTU-13	2013	Public	Packet, Flow	Yes	Real	Yes	81M flows	Yes	No	Botnets (Neris, Rbot, Menti, Virut, NSIS, Murlo, and Sogou)
ECML-PKDD	2007	Public	Packet	Yes	Real	No	46k packets	Yes	No	OScommanding, Xpath injection, Cross-Site scripting, SQL Injection, Path traversal, SSI, and Ldap Injection
ISCX 2012	2012	Public	Packet, Flow	Yes	Emulated	Yes	2M flows	Yes	No	Infiltrating, HTTP DoS, DDoS, SSH Brute Force
ISOT	2010	Public	Packet	Yes	Emulated	Yes	11GB packets	Yes	No	Botnet (Waledac and Storm)
NDSec-1	2016	On request	Packet, Logs	Yes	Emulated	No	3.5M packets	Yes	No	Brute Force, DoS, Web Attack (XSS/SQL injection)
UNSW-NB15	2015	Public	Packet, Other	Yes	Emulated	Yes	2M points	Yes	No	DoS, Generic, Reconnaissance, Exploits, Backdoors, Shellcode, and Worms

attack type. This dataset is publicly available. As it meets our dataset criteria, we use it in our study.

CTU-13: Garcia et.al. [33] created the *CTU-13* dataset at CTU University, in 2011. Attacks are executed using several botnets such as Neris, Rbot, Menti, Virut, NSIS, Murlo, and Sogou. The dataset is public and available in packet format. Instead of labeling anomalous packets to each class of attacks, the authors have divided the traffic into 13 different scenarios. However, contributors have removed the background traffic of normal packets due to privacy and security concerns. As a result, only attack packets are available. We use this dataset in our study in spite of the absence of normal payloads. To handle the lack of normal packets, we borrowed normal packets from CICIDS-2017. The details of the sampling process are discussed in Section V-A.

ECML/PKDD 2007: The dataset was collected from ECML-PKDD [37] competition in 2007. The competition was about analyzing the web traffic to detect or isolate attack patterns. The dataset contains seven different types of attacks. The ECML/PKDD 2007 dataset was generated by recording real traffic. The dataset is publicly available in train and test subsets. It is provided in a packet-based format that contains payloads. As packets of this dataset are labeled, we use this dataset in our study.

ISCX 2012: Shiravi et al. [32] created the ISCX 2012 dataset. The dataset was captured in an emulated network environment. ISCX 2012 contains the traffic of one week. α and β profiles were created to define attack and normal traffic, respectively. The dataset is publicly available in both packet-based and bidirectional flow-based formats. ISCX 2012 contains four types of attacks such as DDoS, SSH Brute Force, Infiltration, and DoS. This dataset contains payloads as the authors have provided packets in pcap files. We use this dataset in our study.

ISOT: Saad et al. [31] published the ISOT dataset in 2010. The malicious traffic was generated from the French chapter of the honeynet project. The dataset is publicly available in packet format. The authors provided the IP addresses of anomalous and normal traffic for labeling. We use this dataset in our study because it meets our criteria for labeled payloads.

NDSec-1: Beer et al. [38] created the NDSec-1 dataset in 2016. The synthetic dataset was captured in a packet-based format. Additional log files are provided by the authors. The attacks present in the dataset are brute force, DoS, Web Attack (XSS/SQL injection). NDSec-1 is available on request. The dataset is labeled and payloads are available. Hence, we use this dataset in our study.

UNSW-NB15: Moustafa et al. [39] created the UNSW-NB15 dataset in a small emulated environment. The network traffic is captured for more than 31 hours. The IXIA Perfect Storm tool was used to create normal and malicious traffic. It contains different types of attacks such as DoS, generic,

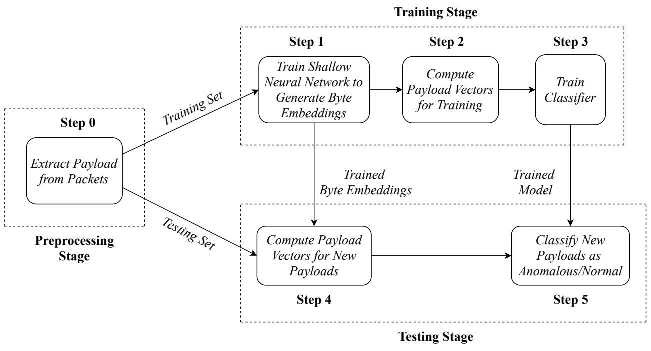

Training Stage
Step 1
Step 2
Step 3
Train Shallow Neural Network to Generate Byte Embeddings
Compute Payload Vectors for Training
Train Classifier
Step 0
Extract Payload from Packets
Preprocessing Stage
Training Set
Testing Set
Trained Byte Embeddings
Trained Model
Compute Payload Vectors for New Payloads
Classify New Payloads as Anomalous/Normal
Step 4
Step 5
Testing Stage

FIGURE 3: Experimental flow diagram of `PayloadEmbeddings`. The flow diagram is divided into three stages, and sequence of flow is identified as steps.

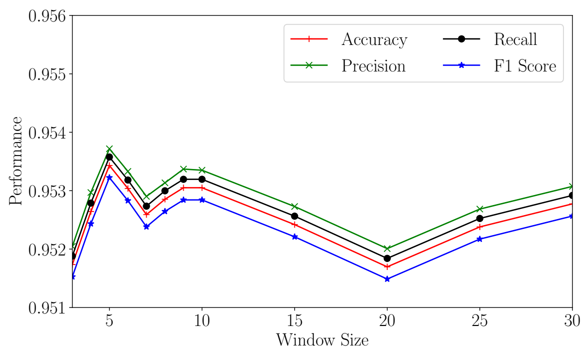


FIGURE 4: Effect of window size `PayloadEmbeddings` on CICIDS-2017 dataset.

exploits, backdoors, reconnaissance, shellcode, and worms. The dataset is publicly available in packet-based and flow-based formats. The authors provided separate train and test sets as part of the dataset. The payload is available in this dataset. We included the UNSW-NB15 dataset in our study.

In summary, we chose the presented ten datasets out of 34 datasets based on whether the dataset is in packet-based format, labeled, and contains anomalous payloads.

V. EXPERIMENTAL DESIGN

In this section, we provide a detailed experimental flow of `PayloadEmbeddings`. We divide our proposed approach into three stages: Data Preprocessing, Training, and Testing stages as depicted in Figure 3.

A. DATA PREPROCESSING STAGE

Step 0. We use ten datasets, including Botnet, CIC DoS, CICIDS-2017, CSIC HTTP 2010, CTU-13, ECML/PKDD 2007, ISCX 2012, ISOT, NDSec-1 and, UNSW-NB15, in our study. First, we extract network packets with payloads. Packets that do not contain any payload data are discarded. Then, each packet is labeled using the ground truth provided within the datasets. We remove the packets with duplicate payloads based on the destination ports and labels. The packet-counts of the resulting datasets are reported in Table 2. By observing the left portion of Table 2, it is noticeable that all datasets are imbalanced before sampling. Imbalanced datasets may cause some classifiers to be biased towards the majority class [40]. To overcome this problem, we perform under-sampling to the majority classes. We randomly remove the instances of the majority classes to balance the attack and normal traffic classes in the datasets. Note that, CTU-13 dataset does not have any normal packets. In order to balance this dataset, we chose normal packets from the CICIDS-2017 dataset. Please note that normal packets used in the CTU-13 dataset and the normal packets used in the CICIDS-2017 dataset are disjoint. The packet-count of each dataset after sampling is reported in the right portion of Table 2. After generating the sample datasets, we divide each of the 10 datasets randomly into two equal parts: the training set and the testing set. The training set is used in *Step 1, Step 2, and Step 3* of the training stage. The testing set is used in *Step 4, and Step 5* of the testing stage.

B. TRAINING STAGE

Step 1. We generate byte embeddings for payloads by training a shallow neural network (already described in Section III-A). We create multiple byte embeddings models for different vector lengths starting from 5 to 50 with intervals of 5 using (1) and (2). During this byte embeddings generation process, we use a window size of 5, which states that for any input byte, its context bytes are located at a distance of 5 or less. The window size is a hyper-parameter and it is chosen after fine-tuning. We selected CICIDS-2017 dataset for hyper-parameter tuning because the dataset has a wide range of attack payloads after sampling (see Table 1 and 2). Figure 4 illustrates the rationale behind using window size 5. All four evaluation metric ranges between 95.1%-95.4%. However, increasing the window size also increases computation time. Hence, we choose window size 5 to keep our approach computationally fast whilst not degrading performance.

Step 2. The next step after byte embeddings generation is to compute feature vectors, *i.e., payload embeddings*, for each payload. For each byte embedding vector length, we compute the *payload embeddings* using (3). The output of this step is multiple *payload embeddings* models.

Step 3. We train k-Nearest Neighbors (kNN) using *payload embeddings* models created in *Step 2*. Note that kNN is non-paramteric. Hence, it uses all the samples of training data for classification in the training step. We experimented with different values of k, in a range between 1-9. Additionally, we performed 10-fold cross-validation for every training dataset. The experimental results are shown in Figures $2-11$ in the supplementary file. Analyzing the figures shows that the performance of `PayloadEmbeddings` slightly increases, decreases, or fluctuates when k is greater than 3. In fact, larger values of k present lower variance, but higher bias. Therefore, we set k to 3 in our experiments.

TABLE 2: Summary of the number of anomalous and normal packets for each dataset.

Dataset	Before Sampling			After Sampling		
	Attack Packets	Normal Packets	Total Packets	Attack Packets	Normal Packets	Total Packets
Botnet	193237	2594419	2787656	45943	45943	91886
CIC DoS	6842	2232433	2239275	6842	6842	13684
CICIDS-2017	76060	76060	152120	76060	76060	152120
CSIC	25065	36000	61065	25065	25065	50130
CTU-13	169040	-	169040	169040	169040	338080
ECML/PKDD	15110	35006	50116	15110	15110	30220
ISCX 2012	109289	1364617	1473906	42600	42600	85200
ISOT	55158	1061591	1116749	55158	55158	110316
NDSec-1	1469	222981	224450	1469	1469	2938
UNSW-NB15	558729	12405175	12963904	47539	47539	95078

C. TESTING STAGE

Step 4. In the testing stage, first, we generate feature vectors, *i.e.,* payload embeddings, for each payload in network packets, similar to the *Step 2*. However, the testing set is used during the *payload embeddings* computation process in this step.

Step 5. Finally, the trained kNN classifier uses the *payload embeddings* created in *Step 4* to classify the payloads as anomalous or normal.

D. IMPLEMENTATION DETAILS

We utilize a multi-core server having 32 processing cores with 2.5 GHz per core and 512 GB of RAM to conduct the experiments of this study. We implement our experiments in Python3.6. To generate the *byte embeddings*, we convert the bytes into integers and use the gensim library. Window size, number of workers, and minimum count parameters are set to 5, 4, and 1, respectively. We use Python's sklearn library for the k-NN classification and cross-validation. To promote reproducibility in science, we share our implementations and supplementary file online[2].

VI. EMPIRICAL VALIDATIONS

In this section, we first present the results of our proposed technique, `PayloadEmbeddings`, across all datasets. Next, we discuss the results via the average byte frequency distributions, confusion matrices, and t-distributed Stochastic Neighbor Embedding (t-SNE) plots.

Figure 5 shows the performance results of our approach over the ten datasets used in this study. For each dataset, the results are reported for different vector lengths ranging from 5 to 50 with intervals of 5. Except for ISOT and NDSec-1 datasets, the performance over the datasets improves when the vector length is increased from 5 to 10. Above 10, the performance of our model either declines or saturates. Our approach achieves its best performance at different vector lengths over different datasets. Botnet, CIC DoS, CICIDS-2017, CSIC HTTP 2010, CTU-13, ECML/PKDD, ISCX-2012, ISOT, NDSec-1, and UNSW-NB15 datasets achieve the best performance (in terms of Accuracy, Precision, Recall and F1-score) for the vector lengths of 25, 50, 10, 45, 25, 40, 15, 10, 5 and, 50, respectively. However, we found in our experiments that there is a sharp rise in the performance when the vector length is increased from 5 to 10 for most of the datasets, except ISOT and NDSec-1. Increasing the vector length from 10 to 50 with intervals of 5, the performance of `PayloadEmbeddings` either improves insignificantly (UNSW NB-15, CIC DoS, ECML/PKDD, CSIC HTTP, CTU-13, and Botnet) or declines (CICIDS-2017, ISOT, NDSec-1, and ISCX2012). Therefore, we use vector length 10 throughout our experiments.

[2]BitBucket: Payload Embeddings

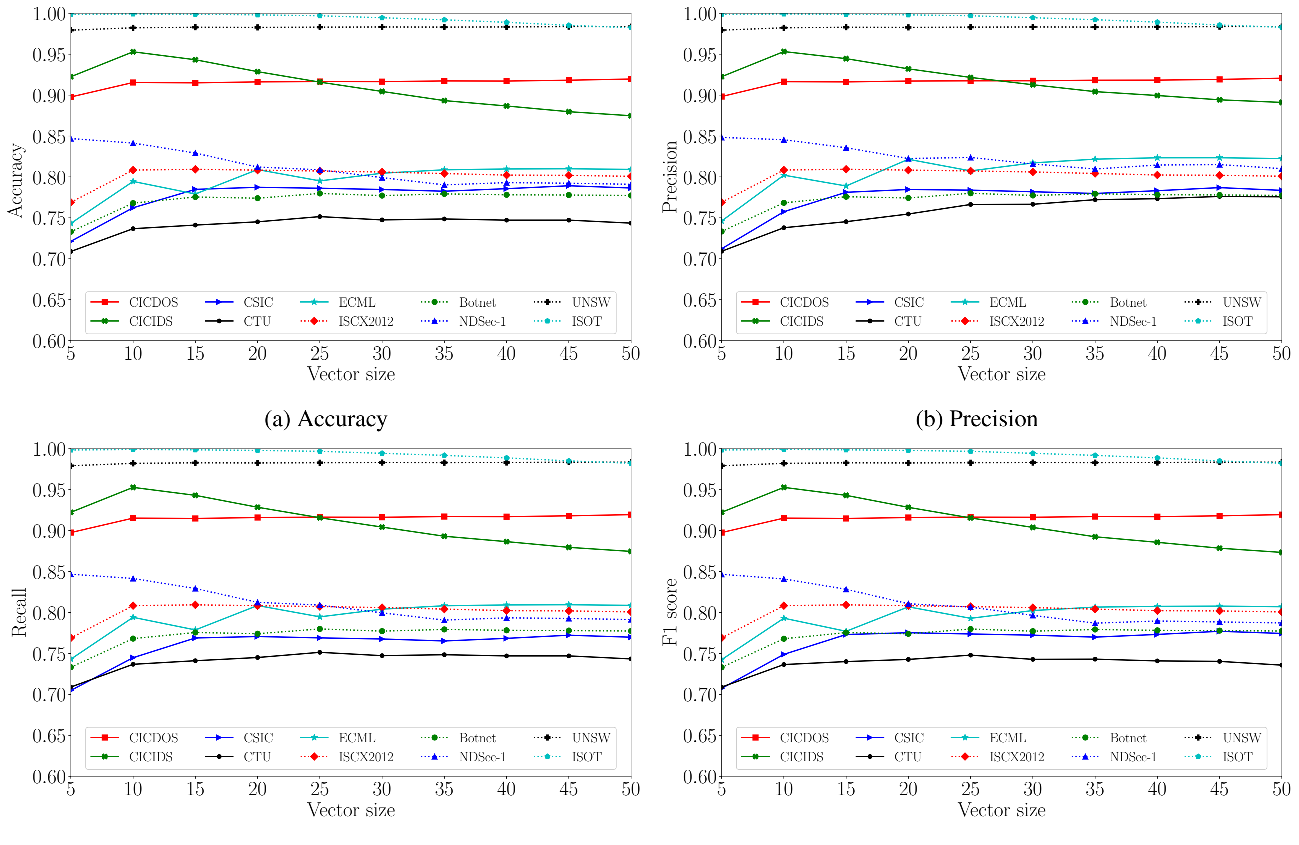
A. PERFORMANCE ANALYSIS

In the following we discuss the performance of our approach over ten datasets. `PayloadEmbeddings` achieves the highest performance on ISOT dataset. The optimal performance is obtained for vector length 10 with accuracy 99%, precision 99%, recall 99% and F1-score 99%. ISOT dataset has only one type of anomaly with payload, *i.e.,* SMTP spam.

TABLE 3: Confusion matrix for ISOT dataset.

Data Type	Normal	SMTP Spam
Normal	**8276**	8
SMTP Spam	19	**8245**

The confusion matrix in Table 3 shows that only 8 out 8284 normal payloads have been misclassified as anomalous and 19 out of 8263 anomalous payloads have been misclassified as normal payloads. To understand the reason behind our high-performance rates over the ISOT dataset, we present the average byte frequencies of anomalous and normal payloads in Figure 6. The frequency of each byte is divided by the total number of bytes in a payload. Then, the average byte frequency distribution is computed for normal and anomalous traffic in the dataset. In Figure 6 it is noticeable that the average byte frequency of anomalous payloads is quite distinguishable from the normal payloads. Normal payloads



(a) Accuracy (b) Precision (c) Recall (d) F1-Score

FIGURE 5: Classification (a) Accuracy, (b) Precision, (c) Recall, and (d) F1-score using k-NN classifier.

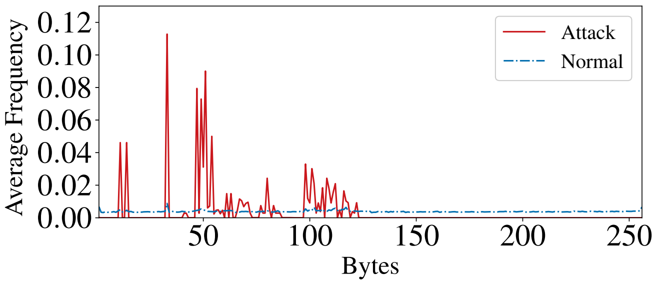


FIGURE 6: Average byte frequency distribution of ISOT dataset.

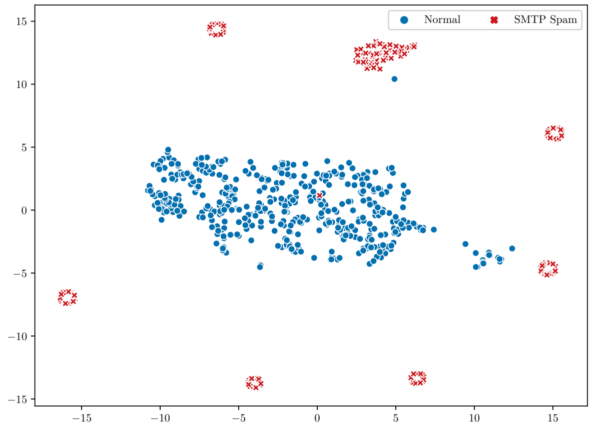


FIGURE 7: t-SNE plot for ISOT dataset.

have uniform-like average byte frequencies, whereas anomalous payloads have fluctuating average byte frequencies. The t-SNE plot for the ISOT dataset is also shown in Figure 7. t-distributed Stochastic Neighbor Embedding (t-SNE) is a very popular dimensionality reduction technique introduced by Maaten and Hinton [41]. t-SNE also attempts to generate representative visualizations of higher dimensional data in two-dimensional space by considering varying, non-linear transformations. For generating t-SNE plots, we selected 500 random samples from each class to produce sample visualizations of clusters. The t-SNE plot in Figure 7 depicts that anomalous payloads are easily separable from normal payloads in ISOT dataset.

Figure 5 shows that amongst the ten datasets, our approach performs the worst on CTU-13 dataset. `PayloadEmbeddings` achieves 75.15% accuracy, 75.13% recall and 74.79% F1-score, when the vector length is 10. The CTU-13 dataset has 13 scenarios generated from different bots such as Neris, Rbot, Virut, Menti, Murlo, Sogou, and

TABLE 4: Confusion matrix for CTU-13 dataset.

Data Type	Normal	Scenario 1	Scenario 2	Scenario 4	Scenario 5	Scenario 6	Scenario 7	Scenario 8	Scenario 9	Scenario 12	Scenario 13
Normal	**6505**	261	60	0	49	0	0	2	65	6	578
Scenario 1	256	**408**	26	0	15	0	0	2	31	3	211
Scenario 2	243	75	**575**	0	8	0	0	2	30	1	134
Scenario 4	0	0	0	**4**	0	0	0	0	0	0	0
Scenario 5	283	47	6	0	**523**	0	0	0	24	1	166
Scenario 6	1	1	2	0	0	**2**	0	0	0	0	1
Scenario 7	0	1	0	0	1	0	**8**	0	0	1	1
Scenario 8	0	0	0	0	0	0	0	**10**	0	0	0
Scenario 9	432	115	32	0	21	0	2	0	**638**	6	216
Scenario 12	3	2	4	0	1	0	0	0	8	**72**	2
Scenario 13	939	282	46	93	0	0	0	0	105	1	**1352**

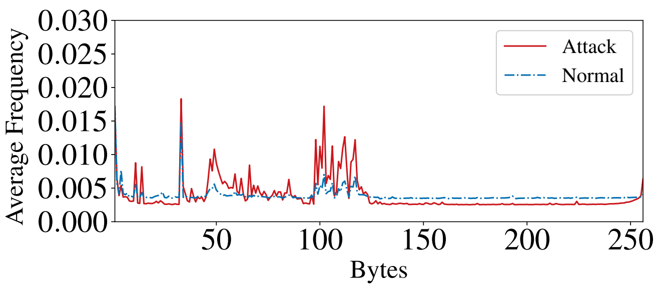


FIGURE 8: Average byte frequency distribution of CTU-13 dataset.

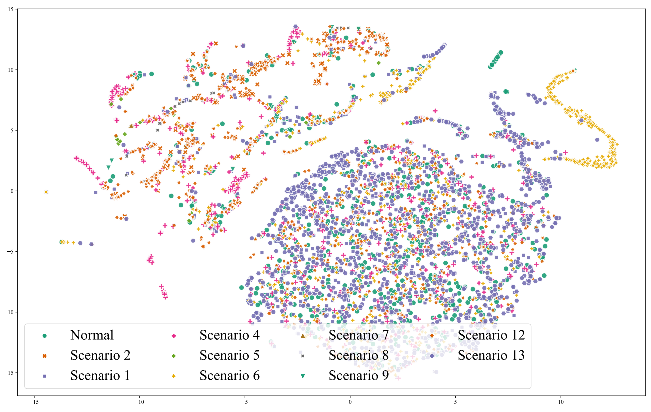


FIGURE 9: t-SNE plot for CTU-13 dataset.

NSIS. Scenarios 1 and 2 include IRC, Spam, and Click-fraud attacks using Neris bot. Scenario 3 includes IRC, Portscans, and US (compiled and controlled by the authors) attacks using Rbot. Scenario 4 includes IRC, DDoS, and US attacks also using Rbot. Scenarios 5 and 13 include Spam, Portscans, and HTTP attacks using Virut. Scenario 6 and 8 include only Portscans attacks using Menti and Murlo, respectively. Scenario 7 includes HTTP attacks using Sogou. Scenario 9 includes IRC, Spam, Click-fraud, and Portscans attacks using Neris. Scenarios 10 and 11 include IRC, DDoS, and US attacks using Rbot. Scenario 12 includes P2P attacks using NSIS. The authors provide trace files and ground truth for each of the scenarios separately. However, the labels only indicate the type of bot used to generate anomalous packets instead of the label of each anomalous class. As a result, the confusion matrix in Table 4 has scenario numbers instead of particular attack classes. Also, anomalous packets from scenarios 3, 10, and 11 do not have payloads. So, we discarded the packets from these scenarios.

By analyzing the confusion matrix in Table 4, we can deduce that our approach can successfully classify anomalous traffic from scenarios 1 and 5 with 100% accuracy. Normal traffic has been classified with 86% accuracy. Anomalous traffic from scenarios 9, 2, 13, and 7 have been mostly misclassified. Scenario 9 has been misclassified as scenario 13 and normal with 22% and 43%, respectively. Scenario 2 has been misclassified as scenario 4 (29%), scenario 13 (14%), 9 (14%), and normal (14%). Scenario 13 and scenario 7 have been the most misclassified as normal traffic with 48% and 30%, respectively. Figures 8 and 9 can explain the higher rates of misclassification that we observe in CTU-13 dataset.

Figure 8 shows that average byte frequencies of normal and anomalous traffic demonstrate similar patterns, which makes it difficult for `PayloadEmbeddings` to differentiate. Figure 9 further explains that the attack classes and normal traffic overlap with each other. It makes it difficult for the classifier to define an optimal decision boundary. As a result, the performance of our approach suffers more.

For other datasets, our approach exhibits varying, but higher performance results. CIC DoS, CICIDS-2017, and UNSW-NB15 datasets have the accuracy, precision, recall, and F1-score with a range between 92%-95% for different vector lengths. CSIC HTTP 2010, Botnet, ISCX-2012, ECML/PKDD, and NDSec-1 datasets have the accuracy, precision, recall, and F1-score with a range between 77%-84% for different vector lengths.

Computational Overhead. In addition to the time complexity of `PayloadEmbeddings` presented in Section III, we present the physical execution times of `PayloadEmbeddings`. We summarize the execution times of `PayloadEmbeddings` and report it in Table 5 for vector length 10 and $k = 3$. Our proposed approach takes 5300 ms to 508000 ms to generate the byte embeddings for different training datasets. The large variation in execution times is due to the number and the size of the payloads in

TABLE 5: Summary of execution time of `PayloadEmbeddings` across 10 datasets when k=3 and vector length=10. The execution times are reported in milliseconds.

Dataset	Training Stage	Testing Stage	
	Byte Embeddings Generation	Feature Computation (per packet)	Classification (per packet)
Botnet	182000	2.09	0.084
CIC DoS	21000	2.97	0.058
CICIDS-2017	355000	2.62	0.111
CSIC	69000	1.83	0.075
CTU-13	508000	3.46	0.105
ECML/PKDD	72000	3.9	0.089
ISCX 2012	177000	2.07	0.089
ISOT	132000	1.09	0.113
NDSec-1	4900	1.43	0.085
UNSW-NB15	178000	2.55	0.075

TABLE 6: Summary of execution time of other state-of-the-art methods along with `PayloadEmbeddings`. The execution times are calculated for CICIDS-2017 dataset and reported in milliseconds.

Methods	Training Stage	Testing Stage (per packet)
PAYL	235,800	1.93
McPAD	451,250	3.48
HMMPayl	634,000	4.5
AEIDS	275,800	2.65
HAST	676,000	4.2
OCPAD	423,450	3.2
EsPADA	1,326,900	7.5
CBID	875,000	9.73
PL-RNN	720,000	2.91
Packet2Vec	515,000	3.54
PayloadEmbeddings	355,000	2.73

the training datasets. Feature computation and classification tasks are carried out in the testing stage. Feature computation refers to computing the `PayloadEmbeddings` from the trained byte embeddings. We compute the average time (in milliseconds) for each payload to generate its corresponding payload embeddings and report it in the feature computation step of the testing stage in Table 5. ISOT dataset is the fastest and ECML/PKDD dataset is the slowest among all the datasets in terms of creating feature vectors for each payload. We employed 10-fold cross-validation to measure and report the performance metrics (as described in Section V-B). On the other hand, we designed a 70%:30% split dataset experiment to measure the average execution time of the classification task per packet. The kNN classifier takes only 0.058 to 0.113 ms per packet to label as anomalous or non-anomalous, on average. In addition, we compute the execution time of other state-of-the art methods based on CICIDS-2017 dataset in Table 6. Training time refers to the time it needs to build a trained model based on the training set of the dataset. Testing time refers to the time it needs to generate features and classify an incoming packet. All the values are reported in milliseconds.

B. COMPARATIVE ANALYSIS

In the following, we compare `PayloadEmbeddings` model with other techniques. We divide these previous studies into categories. They are as follows: Traditional Feature Extraction based, Neural Network based, and other state-of-the-art techniques. First, we present a brief overview of the comparison methods. We also provide the details of their experimental settings. Then, we discuss the performance of `PayloadEmbeddings` against these methods. We summarize the comparative analyses with respect to accuracy, precision, recall, and F1-score over all datasets in Table 7.

1) Traditional Feature Extraction Techniques

We categorize PAYL [4], McPAD [6], and HMMPayl [9] as traditional feature extraction techniques.

PAYL. Wang et al. proposed PAYL [4] that utilizes the byte frequency distributions of normal traffic by applying the n-gram technique. The authors use n-grams (n=1) to produce 256 features that represent the occurrence frequencies of 256 possible bytes in a payload. A centroid model is developed using these 256 features along with their mean and standard deviation. Mahalanobis Distance MAP (MDM) is calculated for new incoming traffic against a precomputed centroid model. The threshold for MDM is set to 256. Incoming traffic is labeled anomalous if the Mahalanobis distance exceeds the threshold.

`PayloadEmbeddings` vs. PAYL: According to Table 7, `PayloadEmbeddings` outperforms PAYL on 8 out of 10 datasets based on accuracy, and F1-score. `PayloadEmbeddings` outperforms PAYL on 7 out of 10 datasets based on precision, and 9 out of 10 datasets based on recall. The main disadvantage of PAYL is that it does not learn context information from payloads, which is captured by `PayloadEmbeddings`.

McPAD. Perdisci et al. proposed McPAD [6] that extracts features from normal payloads using 2_v-gram technique. Instead of using the standard n-gram (n=2) technique, the occurrence frequency of n bytes that are v positions apart is measured. Varying the value of v (0,1,...,10), 11 normal payload models are developed. Test payload is modeled with the same 2_v-gram technique and classified with a one-class SVM classifier for each model.

`PayloadEmbeddings` vs. McPAD: According to Table 7, `PayloadEmbeddings` outperforms McPAD on 9 out of 10 datasets based on accuracy, precision and F1-score. However, in terms of recall, `PayloadEmbeddings` outperforms McPAD on 6 datasets. Such superior performance of `PayloadEmbeddings` over McPAD is attributable to McPAD not explicitly considering the anomalous traffic pat-

TABLE 7: Performance comparison of `PayloadEmbeddings` with other state-of-the-art and traditional methods using kNN.

Methods	Metrics	Botnet	CIC DoS	CICIDS	CSIC HTTP	CTU-13	ECML/PKDD	ISCX-2012	ISOT	NDSec-1	UNSW-NB15
PAYL	Accuracy	0.529 ± 0.003	0.761 ± 0.005	0.701 ± 0.004	0.792 ± 0.001	0.633 ± 0.001	0.495 ± 0.002	0.698 ± 0.003	0.999 ± 0.0	0.735 ± 0.006	0.826 ± 0.004
	Precision	0.873 ± 0.001	0.621 ± 0.002	0.864 ± 0.0	**0.979 ± 0.001**	0.722 ± 0.001	0.706 ± 0.003	**0.937 ± 0.001**	0.998 ± 0.0	0.556 ± 0.009	0.954 ± 0.0
	Recall	0.404 ± 0.002	0.648 ± 0.011	0.532 ± 0.004	0.672 ± 0.001	0.579 ± 0.001	0.366 ± 0.002	0.526 ± 0.003	0.999 ± 0.0	0.836 ± 0.054	0.667 ± 0.006
	F1-score	0.553 ± 0.002	0.634 ± 0.004	0.658 ± 0.003	**0.797 ± 0.001**	0.643 ± 0.001	0.482 ± 0.002	0.674 ± 0.002	0.999 ± 0.0	0.591 ± 0.011	0.785 ± 0.004
McPAD	Accuracy	0.63 ± 0.007	0.757 ± 0.007	0.752 ± 0.009	0.734 ± 0.006	0.788 ± 0.002	0.645 ± 0.006	0.77 ± 0.008	0.914 ± 0.01	0.773 ± 0.002	0.882 ± 0.006
	Precision	0.519 ± 0.007	0.661 ± 0.005	0.823 ± 0.006	0.635 ± 0.009	0.768 ± 0.008	0.518 ± 0.008	0.513 ± 0.004	0.835 ± 0.002	0.755 ± 0.01	0.887 ± 0.006
	Recall	0.774 ± 0.006	0.739 ± 0.004	0.628 ± 0.005	**0.884 ± 0.009**	**0.844 ± 0.006**	**0.898 ± 0.003**	0.615 ± 0.008	0.892 ± 0.004	0.656 ± 0.009	0.807 ± 0.009
	F1-score	0.591 ± 0.005	0.711 ± 0.006	0.699 ± 0.006	0.676 ± 0.004	0.791 ± 0.009	0.637 ± 0.003	0.585 ± 0.003	0.873 ± 0.01	0.708 ± 0.003	0.841 ± 0.008
HMMPayl	Accuracy	0.844 ± 0.009	0.65 ± 0.001	0.922 ± 0.005	0.84 ± 0.007	**0.86 ± 0.002**	0.731 ± 0.006	0.838 ± 0.006	0.889 ± 0.01	**0.889 ± 0.004**	0.803 ± 0.009
	Precision	**0.892 ± 0.008**	0.711 ± 0.001	0.85 ± 0.005	0.778 ± 0.002	0.817 ± 0.004	**0.849 ± 0.005**	0.765 ± 0.004	0.803 ± 0.01	**0.86 ± 0.003**	0.814 ± 0.007
	Recall	0.691 ± 0.01	0.693 ± 0.002	0.909 ± 0.003	0.688 ± 0.005	0.828 ± 0.01	0.717 ± 0.004	0.608 ± 0.004	0.89 ± 0.01	0.792 ± 0.002	0.875 ± 0.01
	F1-score	0.72 ± 0.007	0.701 ± 0.008	0.878 ± 0.008	0.724 ± 0.002	0.821 ± 0.008	0.773 ± 0.005	0.623 ± 0.002	0.858 ± 0.002	0.746 ± 0.004	0.851 ± 0.006
AEIDS	Accuracy	0.821 ± 0.006	0.521 ± 0.009	0.773 ± 0.009	0.845 ± 0.006	0.624 ± 0.006	0.649 ± 0.009	0.756 ± 0.002	0.668 ± 0.002	0.868 ± 0.004	0.733 ± 0.007
	Precision	0.774 ± 0.009	0.525 ± 0.006	0.632 ± 0.008	0.602 ± 0.007	0.803 ± 0.007	0.64 ± 0.004	0.768 ± 0.003	0.654 ± 0.01	0.569 ± 0.004	0.589 ± 0.009
	Recall	0.656 ± 0.001	0.874 ± 0.009	0.584 ± 0.002	0.728 ± 0.004	0.534 ± 0.005	0.506 ± 0.01	0.784 ± 0.008	0.527 ± 0.006	0.827 ± 0.007	0.807 ± 0.004
	F1-score	0.684 ± 0.006	0.66 ± 0.002	0.618 ± 0.009	0.695 ± 0.002	0.702 ± 0.008	0.55 ± 0.004	0.728 ± 0.006	0.618 ± 0.005	**0.899 ± 0.005**	0.693 ± 0.01
HAST	Accuracy	0.805 ± 0.008	0.647 ± 0.002	0.642 ± 0.004	0.615 ± 0.005	0.685 ± 0.007	0.797 ± 0.009	0.878 ± 0.004	0.774 ± 0.009	0.832 ± 0.009	0.768 ± 0.01
	Precision	0.535 ± 0.008	0.617 ± 0.004	0.885 ± 0.006	0.816 ± 0.005	0.731 ± 0.007	0.703 ± 0.003	0.73 ± 0.005	0.756 ± 0.007	0.737 ± 0.01	0.798 ± 0.01
	Recall	0.771 ± 0.008	0.637 ± 0.003	0.689 ± 0.008	0.724 ± 0.009	0.5 ± 0.01	0.792 ± 0.006	0.606 ± 0.002	0.899 ± 0.009	0.802 ± 0.003	0.77 ± 0.006
	F1-score	0.609 ± 0.006	0.613 ± 0.009	0.749 ± 0.002	0.774 ± 0.01	0.679 ± 0.009	0.752 ± 0.003	0.694 ± 0.007	0.793 ± 0.009	0.762 ± 0.007	0.775 ± 0.005
OCPAD	Accuracy	0.694 ± 0.013	0.693 ± 0.005	0.786 ± 0.008	0.78 ± 0.003	0.603 ± 0.009	0.64 ± 0.012	**0.945 ± 0.005**	0.879 ± 0.004	0.767 ± -0.003	0.755 ± 0.016
	Precision	0.632 ± 0.005	0.634 ± 0.009	0.804 ± 0.012	0.636 ± 0.014	0.787 ± 0.009	0.781 ± 0.015	0.813 ± 0.011	0.729 ± 0.005	0.624 ± 0.017	0.863 ± 0.008
	Recall	0.71 ± 0.009	0.686 ± 0.011	0.671 ± 0.007	0.831 ± 0.002	0.624 ± 0.016	0.69 ± 0.01	0.569 ± 0.019	0.914 ± 0.017	0.825 ± 0.013	0.783 ± 0.014
	F1-score	0.599 ± 0.009	0.629 ± 0.01	0.745 ± 0.008	0.788 ± 0.012	0.664 ± 0.006	0.611 ± 0.019	0.644 ± 0.013	0.864 ± 0.017	0.787 ± 0.021	0.766 ± 0.013
EsPADA	Accuracy	0.788 ± 0.012	0.681 ± 0.004	0.644 ± 0.005	0.796 ± 0.015	0.662 ± 0.019	0.681 ± 0.008	0.88 ± 0.014	0.87 ± 0.01	0.795 ± 0.007	0.853 ± 0.005
	Precision	0.769 ± 0.007	0.562 ± 0.013	0.867 ± 0.006	0.75 ± 0.018	0.656 ± 0.012	0.749 ± 0.009	0.687 ± 0.02	0.75 ± 0.011	0.68 ± 0.019	0.752 ± 0.006
	Recall	0.619 ± 0.009	0.644 ± 0.011	0.716 ± 0.012	0.775 ± 0.015	0.649 ± 0.005	0.699 ± 0.002	0.602 ± 0.003	0.881 ± 0.014	0.677 ± 0.006	0.856 ± 0.003
	F1-score	0.563 ± 0.013	0.613 ± 0.011	0.786 ± 0.012	0.657 ± 0.017	0.84 ± 0.005	0.708 ± 0.008	0.63 ± 0.006	0.909 ± 0.002	0.782 ± 0.016	0.681 ± 0.009
CBID	Accuracy	0.676 ± 0.015	0.686 ± 0.005	0.689 ± 0.014	0.722 ± 0.011	0.709 ± 0.003	0.697 ± 0.012	0.717 ± 0.014	0.82 ± 0.009	0.842 ± 0.014	0.828 ± 0.002
	Precision	0.757 ± 0.005	0.604 ± 0.017	0.833 ± 0.015	0.754 ± 0.02	0.764 ± 0.004	0.667 ± 0.009	0.75 ± 0.003	0.8 ± 0.017	0.67 ± 0.011	0.861 ± 0.01
	Recall	0.665 ± 0.008	0.869 ± 0.007	0.735 ± -0.002	0.737 ± 0.01	0.711 ± 0.01	0.606 ± 0.007	0.618 ± 0.009	0.904 ± 0.016	0.718 ± 0.014	0.843 ± 0.009
	F1-score	0.605 ± 0.005	0.702 ± 0.01	0.769 ± 0.015	0.651 ± 0.007	0.746 ± 0.009	0.634 ± 0.01	0.657 ± 0.007	0.841 ± 0.01	0.774 ± 0.008	0.787 ± 0.007
PL-RNN	Accuracy	**0.876 ± 0.015**	0.644 ± 0.014	0.68 ± 0.009	**0.922 ± 0.011**	0.773 ± 0.018	0.562 ± 0.012	0.726 ± 0.007	0.81 ± 0.015	0.862 ± 0.011	0.763 ± 0.007
	Precision	0.822 ± 0.012	0.729 ± 0.01	0.833 ± 0.012	0.715 ± 0.012	0.782 ± 0.009	0.726 ± 0.003	0.685 ± 0.011	0.91 ± 0.007	0.721 ± 0.001	0.72 ± 0.016
	Recall	0.692 ± 0.008	0.748 ± 0.013	0.563 ± 0.006	0.727 ± 0.017	0.723 ± 0.021	0.567 ± 0.007	0.542 ± 0.012	0.751 ± 0.007	**0.898 ± 0.014**	0.882 ± 0.017
	F1-score	0.62 ± 0.006	0.722 ± 0.009	0.693 ± 0.012	0.69 ± 0.012	**0.872 ± 0.015**	0.745 ± 0.004	0.68 ± 0.018	0.782 ± 0.011	0.766 ± 0.013	0.747 ± 0.02
Packet2Vec	Accuracy	0.677 ± 0.017	0.716 ± 0.002	0.662 ± 0.008	0.742 ± 0.024	0.729 ± 0.001	0.667 ± 0.008	0.818 ± 0.007	0.833 ± 0.011	0.739 ± 0.006	0.725 ± 0.003
	Precision	0.713 ± 0.009	0.663 ± 0.005	0.84 ± 0.012	0.719 ± 0.008	**0.822 ± 0.004**	0.846 ± 0.006	0.777 ± 0.008	0.886 ± 0.007	0.692 ± 0.019	0.72 ± 0.011
	Recall	0.678 ± 0.003	0.805 ± 0.003	0.728 ± 0.008	0.819 ± 0.003	0.674 ± 0.007	0.705 ± 0.009	0.594 ± 0.019	0.862 ± 0.011	0.832 ± 0.004	0.879 ± 0.016
	F1-score	0.592 ± 0.011	0.851 ± 0.015	0.627 ± 0.015	0.681 ± 0.006	0.694 ± 0.016	0.606 ± 0.014	0.74 ± 0.021	0.788 ± 0.015	0.739 ± 0.007	0.609 ± 0.014
PayloadEmbeddings	Accuracy	0.78 ± 0.002	**0.92 ± 0.004**	**0.953 ± 0.009**	0.789 ± 0.002	0.752 ± 0.009	**0.81 ± 0.005**	0.809 ± 0.005	**0.999 ± 0.0**	0.847 ± 0.001	**0.984 ± 0.003**
	Precision	0.78 ± 0.008	**0.921 ± 0.008**	**0.953 ± 0.002**	0.787 ± 0.003	0.766 ± 0.001	0.823 ± 0.006	0.809 ± 0.008	**0.999 ± 0.0**	0.848 ± 0.009	**0.984 ± 0.009**
	Recall	**0.78 ± 0.003**	**0.92 ± 0.002**	**0.953 ± 0.006**	0.772 ± 0.005	0.751 ± 0.005	0.809 ± 0.002	**0.809 ± 0.006**	**0.999 ± 0.0**	0.847 ± 0.009	**0.984 ± 0.003**
	F1-score	**0.78 ± 0.005**	**0.92 ± 0.002**	**0.953 ± 0.008**	0.777 ± 0.008	0.748 ± 0.009	**0.808 ± 0.008**	**0.809 ± 0.009**	**0.999 ± 0.0**	0.847 ± 0.003	**0.984 ± 0.005**

terns, alike AEIDS.

HMMPayl. Ariu et al. proposed HMMPayl [9] that also applies the n-gram technique on payloads of normal traffic. A sliding window of n-bytes is used to extract sequences from the payload. A subset of these sequences is selected randomly and passed to a Hidden Markov Model (HMM). The authors set up 5 HMM models. In the training stage, each HMM model assigns a probability to the sequence of normal traffic. In the testing phase, the probability estimation from each HMM model is combined using non-trainable combiners. The authors use four combiners as the maximum, the minimum, the mean, and the geometric mean. The probability scores are combined and checked with a threshold value to classify a payload as anomalous or normal.

`PayloadEmbeddings` vs. HMMPayl: According to Table 7, `PayloadEmbeddings` is only able to outperform HMMPayl on 5 out of 10 datasets based on accuracy, and 6 out of 10 datasets based on precision. However, for other metrics `PayloadEmbeddings` shines against HMMPayl. Our proposed method outperforms HMMPayl on 9 out of 10 datasets based on recall and F1-score. Similar to `PayloadEmbeddings`, HMMPayl considers full payload during the feature extraction process. As a result, HMMPayl achieves better performance in terms of accuracy and precision compared to other methods.

2) **Neural Network-based Techniques** We categorize AEIDS [12], HAST [11], PL-RNN [13], and Packet2Vec [24] as neural network-based techniques.

AEIDS. AEIDS [12] uses a deep learning architecture- Autoencoder to detect low rate attacks as outliers in a dataset. It is an unsupervised approach that is trained on normal traffic. Autoencoders use a neural network to set the target values equal to the inputs. The authors utilize Autoencoder, along with a statistical thresholding approach to identify outliers by generating reconstruction error on the normal traffic. They use a modified z-score to calculate the threshold. Reconstruction error of new incoming traffic is transformed into modified z-score and the targeted traffic is labeled as anomalous if z <3.5.

`PayloadEmbeddings` vs. AEIDS: Our proposed method, `PayloadEmbeddings` outperforms AEIDS on 7 datasets out of 10 based on accuracy. Based on precision and F1-score, `PayloadEmbeddings` outperforms AEIDS on 9 out of 10 datasets. Morever, `PayloadEmbeddings` outperforms AEIDS across all the datasets based on recall. The major reason behind the results is that AEIDS considers only byte frequency distributions to compute reconstruction errors. Additionally, the unsupervised approach does not learn the anomalous traffic pattern in the training phase. Thus, it suffers during testing and/or detection phases.

HAST. HAST [11] uses deep convolutional neural networks (CNNs) to learn low-level spatial features and long short-term memory (LSTM) to learn high-level temporal features from network traffic. HAST has two different architectures. HAST-I architecture takes network flow as input and uses only CNN to learn spatial features. HAST-II architecture takes network packets as input and uses a combination of CNN and LSTM to learn spatial-temporal features. We implemented HAST-II architecture for a fair comparison, as `PayloadEmbeddings` uses packet-based data. In HAST-II architecture, each packet is transformed into a two-dimensional image using One-hot encoding (OHE). For example, if the OHE vector is m-dimensional, then the first n-bytes of a network packet is transformed into an m*n two-dimensional image. The preprocessed data is then fed into HAST-II architecture which produces a vector as output. The softmax classifier is used on the final vector to identify the input traffic to be normal or anomalous.

`PayloadEmbeddings` vs. HAST: According to Table 7, `PayloadEmbeddings` outperforms HAST on 8 out of 10 datasets based on accuracy. `PayloadEmbeddings` outperforms HAST on 9 out of 10 datasets based on precision. Also, `PayloadEmbeddings` outperforms HAST across all datasets based on recall and F1-score. HAST considers only the first 100 bytes of the packets. Therefore, it misses valuable patterns lying in the rest of the payloads. As `PayloadEmbeddings` considers the full length of payloads, it yields better classification performance.

PL-RNN. Liu et al. [13] proposed two deep learning-based models: PL-CNN and PL-RNN. We decided to use the PL-RNN model as it outperforms PL-CNN in their reported results. For the PL-RNN model, the authors employ the LSTM (Long Short-Term Memory) network which consists of 1 hidden layer with 128 hidden states. The first n characters of the payloads are used to train the model. The input size n is set to the average length of the payloads in a particular dataset. Finally, the softmax function is used in the output layer to classify the payloads.

`PayloadEmbeddings` vs. PL-RNN: PL-RNN achieves the highest accuracy on Botnet and CSIC HTTP 2010 datasets. It also achieves the highest recall values on NDSec-1 and F1-score on CTU-13 datasets. However, `PayloadEmbeddings` outperforms PL-RNN based on all the performance metrics across all other datasets. PL-RNN employs an LSTM network that needs fixed-size input. Most of the payloads are either trimmed to 1000 bytes or padded in order to feed a fixed-size input to LSTM. Hence, the network learns from limited or unrelated information which negatively affects the overall performance.

Packet2Vec. Goodman et al. [24] proposed *Packet2Vec* that utilizes a shallow neural network, *i,e., Word2Vec* to generate packet vectors. The authors use n-grams to create a dictionary. As $n = 2$, the vocabulary size reaches to 65,536 (2^{16}). To reduce the complexity, *Packet2Vec* limits vocabulary size to top $|v|$ most frequent n-grams, where $|v|= 50,000$. The vector length for each packet vector is set to 128. Finally, the packet vectors are used as features and classified using Random Forest (RF).

`PayloadEmbeddings` vs. Packet2Vec: Our proposed method outperforms Packet2Vec on 9 out of 10 datasets based on accuracy and recall. Additionally, `PayloadEmbeddings` outperforms Packet2Vec on all datasets based on F1-score, and 8 out of 10 datasets based on precision. However, *Packet2Vec* was able to achieve the highest precision on the CTU-13 dataset. Although Packet2Vec also employs *Word2Vec*, it differs from `PayloadEmbeddings` in vocabulary size and vector length. N-grams ($n = 2$) was not able to capture contextual information from bytes as well as `PayloadEmbeddings`.

3) Other State-of-the-Art Techniques

We categorize CBID [23], EsPADA [22], and OCPAD [7] as other state-of-the-art techniques.

CBID. CBID (Compressed Bitmap Index and traffic Downsampling) [23] is a payload attribution scheme for detecting cybercriminal activities. CBID uses a downsampling technique to store the digest of original traffic in multi-section bloom filters. The downsampling threshold is set to 40 bytes. The bloom filters are optimized with a compressed bitmap index table. If an incoming packet does not match the stored bitmap index table, then it is considered anomalous.

`PayloadEmbeddings` vs. CBID: CBID acheives its highest performance on NDSec-1 dataset, according to Table 7. texttPayloadEmbeddings is able to outperform CBID on all the datasets based on accuracy, precision, recall, and F1-score. CBID utilizes a downsampling technique to deal with the huge amount of network traffic, which makes it lose useful byte information.

EsPADA. EsPADA [22] extracts features from payloads using the n-gram technique where n = 3. The extracted n-grams are stored in Counting Bloom Filters (CBF). Each incoming packet payload is compared to the normal and adversarial models, and a global similarity score (GSC) is computed. If the GSC < τ ($\tau \in [-1, 1]$), the packet is classified as anomalous.

`PayloadEmbeddings` vs. EsPADA: Based on accuracy, EsPADA performs the best on ISCX-2012 dataset among all other datasets. However, `PayloadEmbeddings` outperforms EsPADA on 8 out of 10 datasets based on accuracy. Recall and F1-scores demonstrate that our method performs better than EsPADA on all datasets but one. EsPADA constructs both normal and adversarial models. This method relies on the n-grams of payloads which falls short to capture the contextual relationships between bytes.

OCPAD. OCPAD [7] generates a feature vector for each payload using the n-gram technique where n = 3. Next, the minimum and maximum occurrence probability of n-grams are calculated and stored in a *Probability Tree*. Only normal payloads are used in the training phase. In the testing phase, incoming packets are detected as anomalous if the probability ranges of n-grams are not in the probability tree.

`PayloadEmbeddings` vs. OCPAD: OCPAD achieves the highest accuracy on ISCX-2012 dataset among all the methods. However, `PayloadEmbeddings` outperforms OCPAD on 9 out of 10 datasets based on accuracy, recall, and F1-score. `PayloadEmbeddings` also outperform OCPAD on all but CTU-13 and ISCX-2012 datasets based on precision results. OCPAD uses a Probability Tree constructed from n-grams of normal payloads which falls short on detecting different types of attack patterns correctly.

It is deducible from the comparative analysis that there is no single method that outperforms all other methods over all datasets. Depending on byte distributions, and relevance of contextual information, some methods perform better on some datasets. Nevertheless, `PayloadEmbeddings` outperforms all other techniques on CIC DoS, CICIDS-2017, ISOT, and UNSW-NB15 datasets based on all four evaluation metrics. It also outperforms all other techniques on Botnet, ECML/PKDD, and ISCX-2012 datasets based on F1-score. Hence, `PayloadEmbeddings` can be recommended to use on these seven datasets and achieve a high performance compared to other existing state-of-the-art methods.

VII. LIMITATIONS

`PayloadEmbeddings` considers only the payloads for anomaly detection while ignoring the packet headers. As a result, the proposed method is vulnerable to only header-based attacks such as scanning or probing attacks. On the other hand, it can effectively augment existing header-based intrusion detection systems. In addition, `PayloadEmbeddings` may require to be retrained as the nature of normal and/or attack traffic patterns change in time.

VIII. CONCLUSIONS

In this paper, we propose a payload-based intrusion detection model, `PayloadEmbeddings`, using vector representations of bytes in network packet payloads. The proposed model extends *word2vec* model in the Natural Language Processing domain. We generate payload embeddings from the vector representations of bytes. These embeddings are then fed to a k-Nearest Neighbor (kNN) classifier to detect a network packet as anomalous or non-anomalous. We evaluated our approach over ten different datasets out of 34 datasets that we assessed. Our experimental results show that `PayloadEmbeddings` performs well on ISOT, UNSW-NB15, CICIDS-2017, and CIC DoS datasets with at least 92% accuracy. Our approach also achieves above-75% performance figures on other datasets. Lastly, we compared our approach to ten other state-of-the-art and traditional feature extraction techniques for intrusion detection. We found that no single technique can outperform all techniques on all datasets. In spite of that, we showed our approach outperforms other techniques in terms of accuracy, precision, recall, and F1-score over most of the datasets. Also, we share the implementations of `PayloadEmbeddings` and other comparisons methods online. Moreover, `PayloadEmbeddings` is transferable in other domains where the vector representations of bytes are relevant.

REFERENCES

[1] Y. Bai and H. Kobayashi, "Intrusion Detection Systems: Technology and Development," in *17th International Conference on Advanced Information*

Networking and Applications, 2003. AINA 2003. IEEE, 2003, pp. 710–715.

[2] S. Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy," Citeseer, Tech. Rep., 2000.

[3] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen, "TR-IDS: Anomaly-Based Intrusion Detection through Text-Convolutional Neural Network and Random Forest," *Security and Communication Networks*, vol. 2018, 2018.

[4] Wang, Ke and Stolfo, Salvatore J, "Anomalous Payload-based Network Intrusion Detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2004, pp. 203–222.

[5] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A Content Anomaly Detector Resistant to Mimicry Attack," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2006, pp. 226–248.

[6] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "McPAD: A multiple classifier system for accurate payload-based anomaly detection," *Computer Networks*, vol. 53, no. 6, pp. 864–881, 2009.

[7] M. Swarnkar and N. Hubballi, "OCPAD: One class Naive Bayes classifier for payload based anomaly detection," *Expert Systems with Applications*, vol. 64, pp. 330–339, 2016.

[8] Swarnkar, Mayank and Hubballi, Neminath, "Rangegram: A Novel Payload based Anomaly Detection Technique Against Web Traffic," in *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*. IEEE, 2015, pp. 1–6.

[9] D. Ariu and G. Giacinto, "HMMPayl: An Application of HMM to the Analysis of the HTTP Payload," in *Proceedings of the First Workshop on Applications of Pattern Analysis*. PMLR, 2010, pp. 81–87.

[10] N. Williams, S. Zander, and G. Armitage, "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 5–16, 2006.

[11] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[12] B. A. Pratomo, P. Burnap, and G. Theodorakopoulos, "Unsupervised Approach for Detecting Low Rate Attacks on Network Traffic with Autoencoder," in *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2018, pp. 1–8.

[13] H. Liu, B. Lang, M. Liu, and H. Yan, "CNN and RNN based payload classification methods for attack detection," *Knowledge-Based Systems*, vol. 163, pp. 332–341, 2019.

[14] H. Liu and B. Lang, "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey," *Applied Sciences*, vol. 9, no. 20, p. 4396, 2019.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv preprint arXiv:1301.3781*, 2013.

[16] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[17] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, vol. 86, pp. 147–167, 2019.

[18] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, vol. 2. IEEE, 2000, pp. 12–26.

[19] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, 2000.

[20] J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[21] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. IEEE, 2009, pp. 1–6.

[22] J. M. Vidal, M. A. S. Monge, and S. M. M. Monterrubio, "EsPADA: Enhanced Payload Analyzer for malware Detection robust against Adversarial threats," *Future Generation Computer Systems*, vol. 104, pp. 159–173, 2020.

[23] S. M. Hosseini and A. H. Jahangir, "An Effective Payload Attribution Scheme for Cybercriminal Detection Using Compressed Bitmap Index Tables and Traffic Downsampling," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 850–860, 2017.

[24] Goodman, Eric L and Zimmerman, Chase and Hudson, Corey, "Packet2Vec: Utilizing Word2Vec for Feature Extraction in Packet Data," *arXiv preprint arXiv:2004.14477*, 2020.

[25] S. A. Thorat, A. K. Khandelwal, B. Bruhadeshwar, and K. Kishore, "Payload Content based Network Anomaly Detection," in *2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*. IEEE, 2008, pp. 127–132.

[26] A. Jamdagni, Z. Tan, X. He, P. Nanda, and R. P. Liu, "RePIDS: A multi tier Real-time Payload-based Intrusion Detection System," *Computer Networks*, vol. 57, no. 3, pp. 811–824, 2013.

[27] Y. Luo, S. Cheng, C. Liu, and F. Jiang, "PU Learning in Payload-based Web Anomaly Detection," in *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*. IEEE, 2018, pp. 1–5.

[28] Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *International Conference on Machine Learning*. PMLR, 2014, pp. 1188–1196.

[29] A. Y. Kim, J. G. Ha, H. Choi, and H. Moon, "Automated Text Analysis Based on Skip-Gram Model for Food Evaluation in Predicting Consumer Acceptance," *Computational Intelligence and Neuroscience*, vol. 2018, 2018.

[30] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards Effective Feature Selection in Machine Learning-Based Botnet Detection Approaches," in *2014 IEEE Conference on Communications and Network Security*. IEEE, 2014, pp. 247–255.

[31] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, "Detecting P2P Botnets through Network Behavior Analysis and Machine Leaning," in *2011 Ninth annual international conference on privacy, security and trust*. IEEE, 2011, pp. 174–180.

[32] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[33] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[34] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling," *Computer Networks*, vol. 121, pp. 25–36, 2017.

[35] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." in *ICISSp*, vol. 1, 2018, pp. 108–116.

[36] Information Security Institute of CSIC, *CSIC HTTP 2010 Dataset*, accessed: Nov 20, 2019. [Online]. Available at https://www.isi.csic.es/dataset/.

[37] Raıssi, Chedy and Brissaud, Johan and Dray, Gérard and Poncelet, Pascal and Roche, Mathieu and Teisseire, Maguelonne, "Web Analyzing Traffic Challenge: Description and Results," in *Proceedings of the ECML/PKDD*, 2007, pp. 47–52.

[38] F. Beer, T. Hofer, D. Karimi, and U. Bühler, "A new Attack Composition for Network Security," in *10. DFN-Forum Kommunikationstechnologien*. Gesellschaft für Informatik eV, 2017.

[39] N. Moustafa and J. Slay, "UNSW-NB15: A Comprehensive Data set for Network Intrusion Detection systems (UNSW-NB15 Network Data Set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*. IEEE, 2015, pp. 1–6.

[40] R. Longadge and S. Dongre, "Class Imbalance Problem in Data Mining: Review," *arXiv preprint arXiv:1305.1707*, 2013.

[41] L. Van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

MEHEDI HASSAN is currently pursuing his Ph.D degree in Computer Science at the School of Computing and Informatics, University of Louisiana at Lafayette, LA, USA. He received B.Sc. in Computer Science from the Islamic University of Technology, Bangladesh. He also worked as a Lecturer at Green University of Bangladesh, Bangladesh. His research interests are network security, health informatics and machine learning.

MD ENAMUL HAQUE is currently working as a Research Data Scientist at Stanford University School of Medicine, Palo Alto, CA, USA. He earned his Ph.D. degree in Computer Science at the School of Computing and Informatics, University of Louisiana at Lafayette, LA, USA. He completed M.Sc. in Computer Engineering from King Fahd University of Petroleum and Minerals, Saudi Arabia in 2015. He received B.Sc. in Computer Science from the Islamic University of Technology, Bangladesh. He worked as a Data Scientist at American Family Insurance, Chicago, IL, USA. He also worked as a Software Engineer at Grameenphone Ltd., Bangladesh. His research interest includes complex networks, graph analytics, and machine learning.

MEHMET ENGIN TOZAL is an Associate Professor in the School of Computing and Informatics at the University of Louisiana at Lafayette and a member of the Informatics Program. He received his Ph.D. degree in Computer Science from the University of Texas at Dallas in 2012. His general research areas are complex systems, network security and data/graph analytics. His ongoing research involves health informatics, Internet topology mapping and modeling, Internet security and reliability as well as large-scale graph sampling, summarization and visualization for real-world complex systems.

VIJAY RAGHAVAN is the Alfred and Helen Lamson Endowed Professor in Computer Science at the Center for Advanced Computer Studies. His research interests are in information retrieval and extraction, data and web mining, multimedia retrieval, data integration, and literature-based discovery. He has published around 275 peer-reviewed research papers. These and other research contributions cumulatively accord him an h-index of 37, based on Google Scholar citations to his publications. He has served as a major adviser for 29 doctoral students and has garnered over $13 million in external funding. He brings substantial technical expertise, interdisciplinary collaboration experience, and management skills to his projects. His service work at the university has included coordinating the Louis Stokes-Alliance for Minority Participation (LS-AMP) program since 2001. He has served as a PC chair, PC co-chair or PC member in countless ACM and IEEE conferences. He received the ICDM 2005 Outstanding Service Award. He was an ACM National Lecturer from 1993 to 2006. He was a member of the Advisory Committee of the NSF Computer and Information Science and Engineering directorate, from 2008-2010. He serves on the Executive Committee of the IEEE -TCII and on the Steering Committees of WIC Consortium and Int'l Rough Sets society. He received the WIC 2013 Outstanding Service Award. He is one of the editors-in-chief of the Web Intelligence journal and an associate editor of the ACM Transactions on Internet Technology. He is the founding director of the Visual and Decision Informatics (CVDI), an NSF-funded Industry University Cooperative Research Center, which started its phase II (second 5 years) operations in March 2017, and is a co-director of the Laboratory for Internet Computing.

RAJEEV AGRAWAL is a Computer Scientist in Information Technology Laboratory at Engineer Research and Development Center under the U.S. Army Corps of Engineers, Vicksburg, MS. He is the Data Science lead of the HPC Architecture for Cyber Situational Awareness (HACSAW) Project. He is also a member of HPC-based Deep Learning project team. He is the technical lead of Object Detection and Classification research done under Mobility in Complex Urban Environment (MCUE) project. He is collaborating with ERDC's Geospatial Research Lab (GRL) researchers on Partially Embedded Network Classification and Learning (PENCL) project. Dr. Agrawal is a senior member of the IEEE, a member of the IEEE Computer Society and a senior member of ACM.