2020 7th International Conference on Networking, Systems and Security (NSysS)

22-24 December, 2020, Dhaka, Bangladesh

Networking and Security

Machine Learning Based Malware Detection on Encrypted Traffic: A Comprehensive Performance Study

Onur Barut
University of Massachusetts Lowell
Lowell, MA
Onur_Barut@uml.edu

Matthew Grohotolski
Elizabethtown College
Elizabethtown, PA
grohotolskim@etown.edu

Connor DiLeo
Elizabethtown College
Elizabethtown, PA
dileoc@etown.edu

Yan Luo
University of Massachusetts Lowell
Lowell, MA
Yan_Luo@uml.edu

Peilong Li
Elizabethtown College
Elizabethtown, PA
lip@etown.edu

Tong Zhang
Intel Corporation
Santa Clara, CA
tong.zhang@intel.com

ABSTRACT

The increasing volume of encrypted network traffic yields a clutter for hackers to use encryption to spread their malicious software on the network. We study the problem of detecting TLS-encrypted malware on the client side using metadata and TLS protocol related flow features. We conduct a comprehensive study on a set of widely used machine learning and deep learning algorithms to detect encrypted malware on two malware flows datasets. In addition to reporting the classification accuracy of the approaches under study, we conduct comprehensive experiments to quantify their run-time performance in terms of throughput and system resource utilization such as the CPU and RAM utilization. Moreover, we further boost the speed of the detection systems using acceleration libraries such as DAAL and OpenVINO. Through the quantitative analysis, we provide a comparison on the effectiveness and run-time performance of the machine learning models, and evaluate techniques to accelerate real-world deployment.

CCS CONCEPTS

• **Networks** → **Network monitoring**; • **Security and privacy** → **Intrusion detection systems**; • **Computing methodologies** → **Machine learning algorithms**.

KEYWORDS

malware detection, machine learning, deep learning, encrypted traffic analysis

ACM Reference Format:
Onur Barut, Matthew Grohotolski, Connor DiLeo, Yan Luo, Peilong Li, and Tong Zhang. 2020. Machine Learning Based Malware Detection on Encrypted Traffic: A Comprehensive Performance Study. In *7th International Conference on Networking, Systems and Security (7th NSysS 2020), December 22–24, 2020, Dhaka, Bangladesh.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3428363.3428365

7th NSysS 2020, December 22–24, 2020, Dhaka, Bangladesh

ACM ISBN 978-1-4503-8905-1/20/12…$15.00
https://doi.org/10.1145/3428363.3428365

1 INTRODUCTION

Malicious network traffic affects adversely a large amount of valuable Internet assets, e.g., violating user privacy, stealing sensitive data for ransom, and flooding network devices [23]. With the proliferation of connected devices on the Internet, such malicious network activities generate a tremendous amount of data and call for security and privacy analysis in near real time. As a result, recent studies in Network Traffic Analysis (NTA) start to render security intelligence by mining big network traffic data [9, 20, 30]. NTA, especially from a cybersecurity perspective, has become vital for malware detection, quality of service control, data protection and so on. The growing number of connected devices, increasing network speed and increasingly complex malicious software are just a few factors that exacerbate this situation.

NTA has evolved from port-based approaches to machine learning based techniques over time. Port-based approaches have become unreliable since the modern applications switch to dynamic port allocation. Later, payload analysis, i.e. deep packet inspection (DPI), has become ineffective as the volume of encrypted traffic increases. It is reported that around 95% of Google's traffic is encrypted by June 2020, which was only around 55% six years ago [10]. Moreover, DPI is inefficient for real-time network traffic analysis as inspecting packet payloads cannot keep up with the network speed. Therefore, a lightweight, faster and more accurate malware detection mechanism is in pressing need.

In this paper, we aim to evaluate the candidate machine-learning based designs to resort to an accurate and nearly real-time NTA system for identifying malware in TLS-encrypted traffic. It remains an open question on how machine learning and deep learning models can reliably and efficiently identify malware from encrypted traffic. Flow features are critical for the effectiveness of conventional machine learning models. They are also the inputs to deep learning models due to the randomness of encryption computation, even though a plethora of deep neural network models use raw data (e.g. images) to perform classification in other application domains. In addition, the run-time system performance of these designs must meet stringent throughput requirements for practical deployment.

Our contribution in this paper is three-fold: (1) **dataset**: We are motivated to create a comprehensive encrypted network traffic dataset with more than 100 flow features for benchmarking and future research comparison. Specifically, we obtain the raw traffic

captures from Stratosphere IPS data repository [26] with almost $120k$ encrypted network flow samples belonging to 20 different types of malware and benign classes. We also enrich the dataset for malware detection with network traces obtained from CICIDS2017 dataset [25]. We generate around $75k$ flow samples for 7 different malware types and a benign class; (2) **feature engineering**: We apply network domain knowledge to extract various flow features from the aforementioned dataset, preprocess the features and select the top features to be applied to the analyzing step; (3) **a comprehensive performance study**: We compare a thorough list of machine learning classifiers and deep learning models with the goal of finding the optimal classification accuracy. Distinct from prior work, we also evaluate their model training time, flow prediction time and run-time resource utilization such as CPU utilization and memory consumption. Furthermore, we leverage acceleration libraries to speed up the run-time performance of the systems.

Our experiment results show that complex classification methods such as deep learning algorithms require a substantial amount of system resources, which may not necessarily yield better performance than simpler models. In fact, in the presence of extracted flow features, deep learning models perform less accurate than light-weighted machine learning methods such as logistic regression and random forest. The results also show that the acceleration libraries can significantly speed up the system throughput by up to 68.6 times. These observations provide important insights to real-world deployment.

The rest of the paper is organized as follows. Section §2 introduces the prior arts. Section §3 depicts the way of data collection and §4 the overall design of the proposed NTA system. We evaluate the performance of the system in Section §5. Finally, we conclude the paper in Section §6.

2 RELATED WORK

Random Forest classifier [34] and Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel [1] are two effective and popular methods for network intrusion detection. However, it is difficult to compare one method to another if the researchers use their private data with their own features for traffic classification. For example, Prasse et al. [23] collect their own data to develop a malware detection system based on HTTPS traffic using skip-gram neural language model with LSTM. They compare their method with a random forest classifier using engineered features [4] as input and report that LSTM with neural language model achieves a higher accuracy. Similarly, Anderson et al. [3] use flow metadata, sequence of packet length and times, byte distribution and non-encrypted TLS header information for malware detection in TLS encrypted traffic.

KDD-Cup 99 [16] and NSL-KDD datasets [27] are commonly used network intrusion detection datasets. Several studies have been conducted on those datasets to investigate the ensemble model accuracies such as C4.5 algorithm, decision tree, random forest, SVM and AdaBoost [8, 13, 29]. Similar to ensemble classifiers, different two-level classification algorithms with naive Bayesian as the primary classifier in the first stage, and nominal-binary filtering and cross-validation in the second stage are proposed by Panda et al. [22]. Viyankumar et al. [24] compare various shallow and deep networks to evaluate their effectiveness on intrusion detection using KDD-Cup 99 and NSL-KDD datasets in binary detection and multi-class classification situations, and they find out that deep networks are more accurate than shallow networks.

However, KDD-Cup 99 and NSL-KDD datasets are outdated and do not represent recently emerged attacks. CICIDS2017 dataset [25] is relatively up-to-date and more suitable for detecting current attacks. Authors introduce CICIDS2017 dataset with their results obtained using CICFlowMeter [11] features by comparing several machine learning techniques including k-nearest neighbor (kNN), random forest and ID3 and report best F1 score 0.98 for ID3 algorithm and 0.97 for random forest whose executing time is one-third of ID3.

Gao et al. [9] compare several machine learning and deep learning techniques including Random Forest, SVM and Deep Neural Networks on both NSL-KDD and CICIDS2017 datasets. They perform both binary malware detection and multi-class classification and find out that RF model achieves impressive malware detection accuracy and comparable classification accuracy to other deep learning models in multi-class classification task. Similarly, Haider et al. [12] focuses on DDoS attack detection on software defined networks using CICIDS2017 dataset. They use CICFlowMeter to extract flow features and implement random forest regressor to select important features. Then, they benchmark over different ensemble models and provide 99.45% accuracy using ensemble CNN model.

Marin et al. [21] implement a hybrid model composed of LSTM layer on top of CNN layers to detect malware in the network data obtained from Stratosphere IPS. They conduct packet-level and flow-level analysis and compare the accuracy obtained from raw flow data fed to the proposed model to random forest classifier with around 200 flow features and report that flow-level approach is more accurate and the proposed model achieves 98.6% accuracy while random forest outputs around 97%. Likewise, Yeo et al. [32] use public data from Stratosphere IPS to classify 5 different malware types and a benign class. They compare several machine learning algorithms along with a CNN model on top of 35 flow features extracted using Netmate [2] and conclude that CNN model with random forest classifier achieves superior classification accuracy.

Researches mentioned above have one limitation in common: they only focus on the classification accuracy but ignore the computational burden of the proposed methods. However, there are also other researches that consider and discuss about the computational costs of the proposed classification methods and attempt to speed up the performance. Motivated by the characteristics of machine learning based classification approaches, Luo et al. propose a classification tree search method called explicit range search to accelerate traffic classification on FPGA devices [19]. Kim et al. [17] implement a LSTM based model which achieves 96.93% accuracy on the KDD Cup 1999 dataset, using an Intel i7 as CPU and GTX Titan X as GPU for acceleration. Similarly, Yin et al. [33] construct an RNN based model for both binary detection and multi-class classification, achieving 83.28% accuracy on NSL-KDD dataset in 5, 516 seconds. On the other side, Liu et al. [18] propose a CNN based model for NIDS and obtain 97.7% accuracy on KDD Cup 1999 dataset with GPU acceleration GTX 1080, 32GB RAM with OS Ubuntu 14.04 in experimental setup.

To the best of our knowledge including the above mentioned literature work, we observe that deep learning methods are widely proposed to achieve state-of-the-art results; however, most of the prior researchers either focus on a narrow spectrum of malware type classification or do not consider the time to train those models and time to execute for a single instance detection. In real-world deployment where the network appliance may have very limited computing and memory resources, it is crucial that the malware detection system is fast, accurate and has a small footprint of resource utilization. In this regard, we compare several machine learning and deep learning approaches in this study in terms of accuracy, time to train the model, time to classify a single flow instance, and utilization of CPU and memory using two different network intrusion detection datasets. At the end, we propose a fast and accurate malware detection model using accelerated tools.

3 DATA COLLECTION

3.1 Dataset Preperation for TLS-encrypted Malware Detection

Stratosphere IPS [26] and Canadian Institute for Cybersecurity (CIC) [5] are two publicly available sources for network capture data. Both CIC and Stratosphere IPS provide several different types of raw network packet captures for Intrusion Detection Systems (IDS) in '.pcap' format. However, pcap files are not readily leverageable by most data analytics libraries and the provided malware traces are too scattered across different files.

Therefore, we firstly create a new set of network trace dataset from Stratosphere IPS by extracting 20 different types of malware classes such as Adload, PUA, TrickBot, Ramnit, and Ransom etc. along with benign traffic data from different pcap files mostly captured in 2017 and name it as "**dataset1**". Since we only focus on binary malware detection in this study, all of the different types of malware traces are labelled as "malware" and all normal traces are labelled as "benign".

Then we use CICIDS2017 [25] dataset from the CIC repository to generate our "**dataset2**". In CICIDS2017, raw capture data contains the whole trace record throughout the day. However, during the data collection process, malware attacks are performed only in a specific time slot of the day. Therefore, we have to filter those network traffic of our interest according to the time stamp [6], then label the extracted data into either "malware" or "benign". Similar to "dataset1", we also use binary labels for CICIDS2017 regardless of their malware types.

3.2 Preprocessing and Feature Extraction

After obtaining "dataset1" and "dataset2", the captured raw data are fed to our modified version of feature extractor derived from Cisco Joy [7] to extract more than 200 flow features. We aim to render a comprehensive list of feature representations including metadata, TLS, DNS and HTTP features. The detailed data preprocessing, feature extraction and labelling methods are described in the following subsections and depicted in Figure 1. We also anonymize the data by masking the source and destination IP addresses. This also eliminates the bias introduced by the data collection setup due to the static IP addresses of the computers used to capture the network flows.

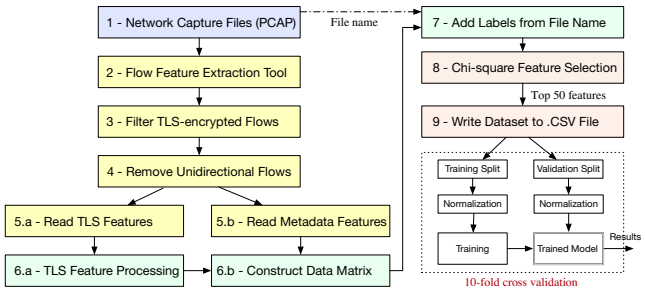

Figure 1: Proposed approach and preprocessing steps for encrypted malware detection

3.3 Data Preprocessing

When pcap files come in, the flow feature extractor first assembles long network flows from both inbound and outbound directions into data records (rows) with length capped at 200 packets. Flows less than 200 in length are kept intact. Extracted features are stored in JSON format. Metadata features such as number of packets, time duration of the flow, source port, destination port etc. can be extracted for every flow instance because these features do not rely on any protocol. However, protocol-specific features such as TLS, DNS and HTTP can only be extracted if the flow instance contains packets utilizing the given protocols. Therefore, only a small portion of the flows contain protocol-specific features while all of the flows contain metadata features. After extracting metadata, TLS, DNS, and HTTP features, we de-identify the data by masking out the source and destination IP addresses. Doing so eliminates the bias introduced by the static IP addresses of the hosts used to capture the network flows. Later, we remove *time_start* and *time_end* and substitute in *time_length* as a new feature which is obtained by subtracting *time_start* from *time_end*. This step also helps remove the time stamp our models could leverage to make biased prediction. After that, "id" and "label" for each flow are added to output the feature files in JSON format. Finally, we filter the TLS encrypted flows by checking if there is any TLS-encrypted feature available in the flow. This step ensures both our datasets are TLS-encrypted.

We observe that there are many flows who have zero inbound packet features. In other words, the flows are unidirectional, or unacknowledged. Since our proposed NTA system is going to be deployed on the client side of the network, it makes sense to discard the all the unacknowledged queries. Therefore, we filter out unidirectional flows and keep around 114, 000 and 75, 000 encrypted bidirectional flows in "dataset1" and "dataset2", respectively.

3.4 TLS feature processing

Machine learning models are often trained with a two dimensional matrix dataset whose rows represent flow instances and columns the feature values. Metadata features consist of statistics and numerical representations for flow properties that can be computed for any flow. For example, source port, destination port, number of inbound packets, number of outbound packets and time duration of the flow are some of the metadata features that can be extracted for every flow. Meanwhile, TLS features not only contain numerical features such as number of ciphersuites offered by client and server,

number of TLS encrypted packets in the flow, etc., but also variable-sized non-numerical feature arrays such as the ciphersuites offered by the client, supported TLS extensions of the client, etc. Therefore, these variable-sized non-numerical features must be converted into fixed length numerical features in order to use the TLS information along with the metadata features in the input data matrix.

First of all, TLS features need to be loaded into the data matrix along with metadata features. Tls_cnt, tls_cs_cnt, tls_ext_cnt, tl_key_exchange_len, tls_svr_cnt, tls_svr_cs_cnt, tls_svr_ext_cnt and tls_svr_key_exchange_len features can be easily loaded into data matrix since each of them is represented by a single integer value. Therefore, each of these features will occupy a single column in the data matrix.

Tls_cs and tls_ext_types features have variable sizes for each flow. For instance, the tls_cs feature for a single flow instance may contain '0039', 'c009' and other ciphersuites as an array of strings. Tls_ext_types is also similar to tls_cs. Hence, those ciphersuites and extension types need to be analyzed and top-N most common features, say N=10 for each class in the dataset must be obtained. Usually, the number of common features is larger than 10 for a multi-class classification dataset, for example, 13, since different classes may contain different top common ciphersuites. Therefore, we allocate separate 13 columns in the data matrix for these ciphersuites and if the flow instance contains any of those ciphersuites in its tls_cs feature, then 1 is assigned for the corresponding column. If not, 0 value is assigned. At the end of the columns, we add another column as the 14^{th} column which represents the ciphersuites that are not among top commons. For example, if a flow instance contains 4 other ciphersuites that are not among top commons, then 4 is assigned to the 14^{th} column of tls_cs feature in the data matrix for that flow instance. The same approach applies for the tls_ext_types feature.

Tls_svr_cs feature is a single-valued string type feature with a ciphersuite selected among the advertised ciphersuites in tls_cs. Therefore, we have a binary value for this feature. If the selected ciphersuite is among the common-N ciphersuites in tls_svr_cs, then 1 is assigned to tls_svr_cs, otherwise 0 is assigned to the corresponding column in the data matrix.

Tls_svr_ext_types has the same property with the tls_cs and tls_ext_types. So, the same procedure also applies to this feature as well.

Tls_len and tls_svr_len features are variable length integer arrays with payload sizes. For instance, assuming tls_len has [213, 44, 64, 32] and tls_svr_len has [76, 35, 114] as feature values, we allocate 4 columns for each to represent this information in the data matrix: (1) length of the array, (2) minimum value in array, (3), maximum value in array, (4) mean value of array.

3.5 Feature Selection with Chi-square Algorithm

Selecting important but smaller number of features plays a crucial role while training a machine learning classifier. A good set of selected features not only reduces dimensionality of the dataset and hence decreasing the computational cost but also boosts the accuracy of the classifier by reducing overfitting. Chi-square test is also widely used as feature selection method in many prior studies and also implemented in network traffic analysis [31] and intrusion detection systems [28]. Chi-square is a numerical test which determines the dependency of each feature to target variable, which is the label of the flow instance. If the target variable has a weak dependency to the selected feature, then it can considered as irrelevant. Hence, the features can be sorted by their relevance to the target variable and highly relevant subsets of the whole features can be selected for the proposed classification model.

4 SYSTEM DESIGN

4.1 Classifiers Under Study

In this study, TLS-encrypted malware flows are detected from the network traffic captures using a modified version of `Joy` to extract the proposed features in Section §3. Firstly, chi-square test is applied to the dataset and the first 50 important features are selected and stored in a separate csv file. Then, the data are standardized, i.e. mean is subtracted and each feature is divided by the standard deviation. Finally, several machine learning models and deep learning models are evaluated and compared in terms of performance and resource utilization on CPU and GPU, respectively.

Table 1 shows the list of top-50 features selected by chi-square for both datasets used in this study. Some histogram-like features such as *pld_ccnt* are represented by a fixed size of arrays. Feature selection algorithm may only select a few indices for that feature and the numbers inside brackets stand for the index values that are selected by the chi-square algorithm. The superscript * indicates that the selected feature or index of the histogram-like feature is exclusive for "**dataset1**". Similarly, superscript $^+$ indicates exclusiveness for "**dataset2**". Other features that do not have a superscript are shared between both datasets.

4.1.1 Logistic Regression. Logistic regression model is a supervised classification model and can be described as a parametric estimator that is computed by taking the sigmoid of the linear combination of the input variables. It outputs the probability of the given sample belonging to the specified label. Neither standard library nor the accelerated library has a parameter to train a logistic regression model.

4.1.2 Random Forest. Random forest classifier is another example for a supervised classification methods. It consists of several decision trees and outputs an ensembled prediction by voting. The number of estimators are set to be 100 and maximum depth of a decision tree is set to be 10 in this study.

4.1.3 Support Vector Machines. The purpose of SVM is to set a hyper-plane which separates the two support vectors where support vectors are the closest sample point of a class to decision boundary as far as possible. Design parameters for SVM classifier which are regularization parameter C and the kernel function used are set to be 1.0 and linear kernel, respectively.

4.1.4 k-Nearest Neighborhood. kNN classifies a new instance by comparing its distance to the k closest instances and assigns the label of those instances that are majority. In this study, k is set to be 5 and Minkowski distance whose formula is given in equation (1) is used to compute the distance.

Table 1: Description of extracted flow features

Feature & Description
Metadata Features
src_port: source port
dst_port: destination port
bytes_out: total bytes out
num_pkts_out*: total packets out
bytes_in: total bytes in
num_pkts_in: total packets in
time_length: time duration of the flow
intervals_ccnt[]: compact hist. of pkt arriving intervals [0*,1*,2]
ack_psh_rst_syn_fin_cnt[]: histogram of tcp flag counting [0*,1*,2+]
hdr_bin_40: # of pkts with header lengths between 28 and 40
hdr_ccnt[]: compact histogram of header lengths [1,3+,4,6+,7+,8+]
hdr_mean+: mean value of header lengths
pld_ccnt[]: compact histogram of payload lengths [0*,1+,3+]
pld_max: max value of payload length
pld_mean+: mean value of payload length
pld_median: medium value of payload length
rev_...: flow features of the reverse flow
rev_ack_psh_rst_syn_fin_cnt[]: [0,1]
rev_hdr_ccnt[]: [1,2*,4+,6+]
rev_intervals_ccnt[]: [0,1*,2]
rev_hdr_pld_128*:
rev_pld_ccnt[]: [0*,1*,3+,15]
rev_pld_distinct*: # of distinct values of payload length of rev. flow
rev_pld_max:
rev_pld_mean:
rev_pld_var:
TLS Features
tls_cnt*: # of tls packets
tls_cs_cnt+: # of ciphersuites
tls_len[]: array of tls payload length [len*, min, max, mean]
tls_cs[]: array of ciphersuites value [0039*,c009+,c02f*,cca9+,other+]
tls_ext_cnt: # of tls extensions
tls_ext_types[]: array of tls extensions [0010,0017+,ff01*,ff03+]
tls_key_exchange_len: length of tls key exchange
tls_svr_...: tls features advertised by the server
tls_svr_cnt*: # of ciphersuites advertised by the server
tls_svr_ext_cnt: # of extensions advertised by the server
tls_ext_types[]+: array of tls extensions [0010]
tls_svr_len[]: array of tls payload length [len*, min, max, mean]

*: "dataset1" exclusive features
+: "dataset2" exclusive features

$$Minkowski\_Distance = (\sum_{i=1}^{N} |x_i - y_i|^k)^{\frac{1}{k}} \quad (1)$$

where N is the number of flow instances in the training set and k stands for the number of closest instances whose distances will be measured to classify a new instance.

4.1.5 Multi-Layer Perceptron. The simplest MLP classifier contains an input layer, a hidden layer, and output layer. In this study, MLP classifier is created with a single hidden layer. In the hidden layer, 128 perceptron units are implemented. L2 regularization factor *alpha* is set to be 0.0001 and the model is trained using *adam* as the optimizer.

4.1.6 Convolutional Neural Networks. Since CNN produces state-of-the-art results for image classification and text classification, both 1-D CNN and 2-D CNN models are implemented for binary malware detection problem where the extracted flow features are fed to CNN as input data. For the two dimensional CNN model, the input data is reshaped from 50x1 to 10x5. For both of the CNN models, kernel size is chosen to be 3 with stride 1. 128 filters are used for each convolutional layer and 128 hidden units are implemented for fully connected layers. Two CNN layers and two fully connected layers are used to construct CNN models. The dropout ratio is set to 0.5 to prevent overfitting. The model architecture for both CNN models are given on the top of Figure 2.

4.1.7 Long Short-Term Memory Neural Network. LSTM layers encode a representation of the given sequential data by exploiting the temporal information. Therefore, an LSTM layer requires a two dimensional input data $\mathbf{X} = [\mathbf{x}_1{}^D, \mathbf{x}_2{}^D, ..., \mathbf{x}_n{}^D]$ whose dimensions are nxD where n is set to be 10 and D is 5. Two LSTM layers followed by the output softmax layer are used to construct a Recurrent Neural Network (RNN) with number of hidden units set to be 128. Similar to CNN models, dropout ratio is chosen to be 0.5 The model architecture is given on the bottom left of Figure 2.

4.1.8 CNN+LSTM Hybrid Neural Network. Several hybrid classifiers are proposed and proven to be successful for network intrusion detection systems [21]. These classifiers use CNN layers in the earlier stages of the model to exploit spatial information and LSTM layers in the later stages to learn temporal information from the dataset. Therefore, the CNN+LSTM hybrid model proposed in [21] is also implemented for our experiments with the modified design parameters for our datasets. For instance, kernel size is set to 4 with stride 1. 32 filters are used for each convolutional layer and 200 hidden units are implemented for LSTM layer and fully connected layers. Two CNN layers followed by an LSTM layer and two fully connected layers are used to construct hybrid model. Similar to previous deep models, the dropout ratio is set to 0.5. The hybrid CNN+LSTM model architecture is given on the bottom right of Figure 2.

For all of the deep learning models, hyperparameters are optimized by a manual search around default values. For example, learning rate is set to 0.001, decay rate of learning rate to 0.00001, batch size to 100, and regularization parameter for the output layer to 0.0001 to prevent overfitting. All models are trained for 100 epochs using Adam optimizer.

4.2 Model Implementation and Acceleration

To evaluate the proposed approaches, we prepare two versions of machine learning/deep learning models for comparison: 1) the plain version and 2) the accelerated version. The same model parameters are used for both versions to guarantee a fair comparison across the entire collection.

In the plain version, we utilize Scikit-learn library for traditional machine learning classifier on CPU and Keras library with Tensorflow backend for deep learning models on GPU. In the accelerated version, we aim to accelerate model training and inference on CPUs. The Intel Data Analytics Acceleration Library (DAAL) [14] provides optimized machine learning routines and algorithms targeted for

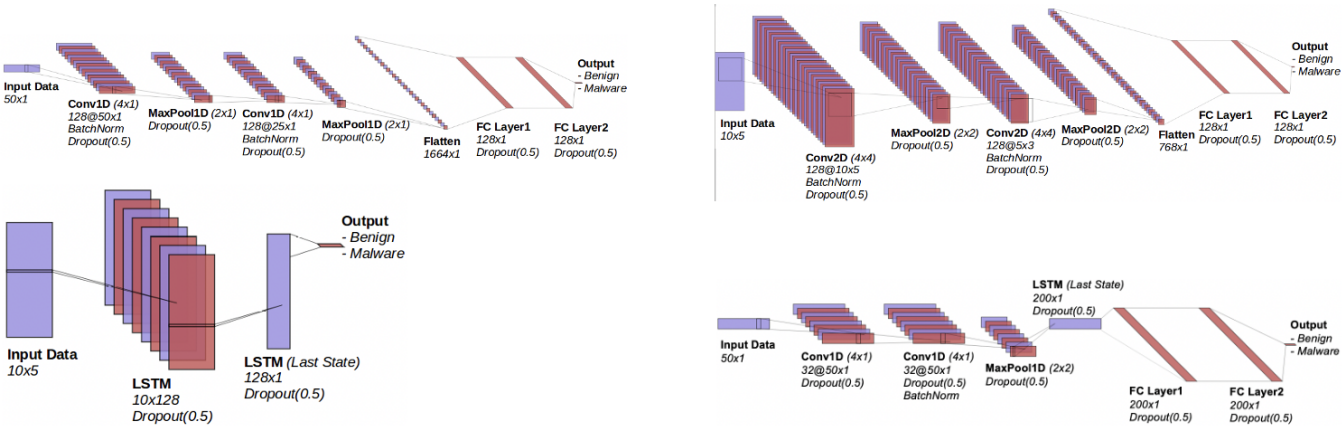

Figure 2: Proposed deep learning model architectures. Top: CNN models (1D and 2D), Bottom Left: LSTM model, Bottom Right: CNN+LSTM hybrid model

Table 2: Experiment Platform Specification

Item	Specifications
Op. Sys.	Ubuntu 16.04.6 LTS
CPU	2x Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz
GPU	2x GV100GL NVIDIA Tesla V100 16 GB
RAM	64 GB DDR4 @ 2666 MHz
Intel DAAL	version 2020.1
Intel OpenVINO	version 2020.3

Intel processors. DAAL functions maximize processing speed by leveraging the instruction set, vector width, core counts, and memory architecture for each processor they run on. Similar to the popular Scikit-learn library, DAAL offers support for many of the popular machine learning algorithms. For our purposes, we provide DAAL accelerated logistic regression, random forest, support vector machines, and k-nearest neighbors to compare with their plain counterparts. The deep learning models are also accelerated by using Intel's OpenVINO library [15]. OpenVINO consists of two main components, its Model Optimizer and Inference Engine. The Model Optimizer is responsible for taking pre-trained models from our plain neural network models such as ANN, CNN, LSTM and CNN+LSTM hybrid models; it then transforms these models into a Intermediate Representation which OpenVINO can utilize to drastically increase the inference rate. After the model is transformed, we utilize OpenVINO's Inference Engine to yield the final inference. We implemented the proposed models on a machine with the hardware and software specifications given in Table 2.

4.3 Evaluation Metrics

Classification Effectiveness: To compute the performance of the proposed binary malware detection systems, accuracy in the validation split (ACC), detection rate as true positive rate (TPR), and false alarm rate (FAR) are obtained using the equations (2) to (3). Dividing the throughput value of the accelerated model to the standard counterpart gives the performance ratio (PR) as presented in equation (4).

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

$$TPR = \frac{TP}{TP + FN}, FAR = \frac{FP}{TN + FP} \tag{3}$$

$$PR = \frac{t_a}{t_s} \tag{4}$$

where TP, TN, FP, FN, t_a, and t_s stand for true positive, true negative, false positive, false negative, throughput of accelerated model, and throughput of standard model, respectively.

Run-time Performance and Resource Utilization: In order to measure the time to train and time to perform prediction, time module of python is used. Time intervals measured for training and inference stages represents for only the time taken to complete the specified stage. In other words, those times do not contain the time to read the data or preprocess the data; they purely represent the computation time. Using the measured computation time, we then report the throughput in units of flows per second.

In addition to those metrics, CPU utilization is obtained from psutil library of python and memory consumption is observed using memory_profiler module. For the GPU implementation, we report the GPU memory utilization. All results obtained from these metrics are averaged using 10-fold cross-validation.

5 RESULTS ON ENCRYPTED MALWARE DETECTION

5.1 Performance results with plain ML

Several machine learning classifiers are trained on "dataset1" and the performance measurements for this dataset on different models are provided in Table 3. Although validation accuracy (ACC) is also provided, true positive rate (TPR) and false alarm rate (FAR) are more important metrics, especially when the dataset is imbalanced. Therefore, in terms of correct classification ability, random forest classifier achieves the highest detection rate 99.996% and lowest false alarm rate 0.0297%.

Throughput of the classifier in terms of flow per second for training and prediction stages are given in `t_tr` and `t_pr` columns,

respectively. They are calculated by dividing the number of flow from the given dataset split to the time interval of the stage. From the system resource utilization perspective, we observe from Table 3 that logistic regression classifier is the fastest on prediction stage and it has the lowest cost of average CPU utilization 167% and 489 MB memory for the training stage. However, logistic regression yields the higher false alarm rate, which is undesirable for a network traffic analysis system.

Table 3 also indicates how the different models are implemented in the selected standard library. For example, average CPU utilization for kNN classifier is 100%, which means that this algorithm uses a single core throughout the training stage. However, SVM, whose detection rate is second best after RF, utilizes 109% of CPU that indicates the model is rarely requiring a second core to complete the training for the given throughput rate. Similarly, the most accurate model which is random forest classifier requires more than 10 cores to achieve such training and classification throughput while MLP classifier is the most CPU greedy one requiring more than 20 cores.

Different machine learning models require various CPU and memory utilization and outputs similar accuracy while training and classification speeds vary a lot due to the nature and the implementation of the algorithm. For a network traffic analysis system, correctly identifying malware traffic and producing less false alarm are the most critical security indicators. Moreover, detection speed is also an important factor especially when deployed on a resource limited edge network. In this scenario, CPU and memory utilization are limited. E.g., Table 3 shows that random forest model is highly accurate and relatively faster in the prediction stage if the host device is capable of providing 10 or more cores. However, for a dual-core or quad-core edge device, radom forest may not yield the desiring accuracy and speed. In conclusion, tradeoff has to be made when deciding on a classification model for encrypted malware traffic detection.

5.2 Performance results with plain DL

Different deep learning models are implemented for our encrypted malware detection system. Similar to the results obtained from machine learning classifiers, Table 4 demonstrates the comparison between various DL models on "dataset1". Throughput values are calculated as explained for machine learning models. Training throughput values are obtained while training the models on GPU while prediction throughput is obtained on CPU. Additionally, GPU memory utilization is also provided since the proposed deep learning models are implemented on GPU. From this table, we observe that MLP model is the most accurate deep learning model with TPR 99.995 and FAR 0.1305. Furthermore, MLP requires the lowest memory as 860 MB from RAM and it is the fastest model both in training and prediction stages with 710.6 and 92252 flow per second, respectively. The hybrid CNN+LSTM model has the highest false alarm rate with 2.6824% which is significantly worse than MLP model with 0.1305% false alarm rate. A common observation for all DL models is that all of the proposed DL models require 7 to 9 cores on average for training process and 1.3 to 1.7 GB memory from RAM even though most of the computation is handled by GPU. An exception to that is for MLP model whose CPU core utilization is 1.77 on average with 860 MB memory requirement from RAM. We also observe the memory allocated from GPU using NVIDIA's System Management Interface (nvidia-smi) and provided as the right-most column of Table 4. The utility shows that 305 MB memory per GPU is allocated to training stage for all of the DL models.

Interestingly, we notice that the best performing DL model (LSTM based RNN) is less accurate than RF model, which yields the best accuracy for "dataset1" among all ML classifiers. For example, detection rate for LSTM model is 99.964% which is smaller than RF model whose detection rate is 99.996%. Similarly, LSTM model produces a higher false alarm with 0.5518% while RF model has 0.0297% false alarm rate. Therefore, it can be concluded that an accurate encrypted malware detection system could be achieved by engineering flow features and carefully selecting the most important 50 of them and implementing a random forest classification model. Another reason why DL models are highly accurate but cannot achieve state-of-the-art for this problem is that the dimensions of the input data which are 50x1 for 1D CNN and 10x5 for 2D CNN and LSTM models, are relatively small for a DL model so that a simpler traditional machine learning models such as random forest can achieve a higher accuracy.

5.3 Performance results with DAAL-accelerated ML

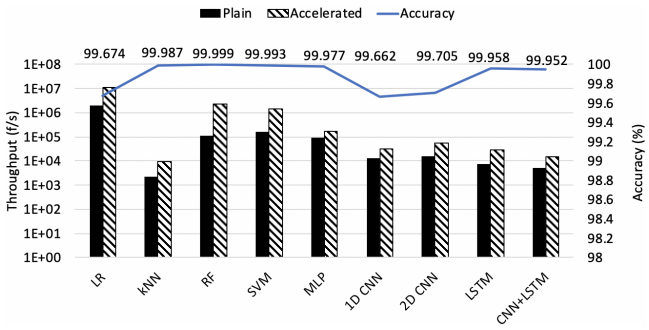

Figure 3: Performance comparison of accelerated classifiers on "dataset1"

The machine learning classifiers implemented using standard library are also implemented using an accelerated library. Table 5 depicts the results obtained using an accelerated library and Figure 3 gives the throughput comparison of plain and accelerated classifiers on the prediction stage along with their accuracy. Since the implementation of a classifier on standard sklearn library and accelerated library (DAAL) might differ, we observe slightly different accuracy, detection rate and false alarm rate values. For example, logistic regression with sklearn utilizes 1.67 cores on average while the accelerated counterpart utilizes 6.87 core on average. Additionally, accelerated logistic regression model is less accurate; however, it consumes less amount of memory and 5.6 times faster in the prediction stage and 1.33 times faster in the training stage. On the other hand, for kNN classifier, accelerated library achieves higher accuracy with significant acceleration in both training and prediction stages and requires less amount of memory. Unlike the

Table 3: Performance output of ML classifiers with standard library (sklearn) on "dataset1"

Model	ACC (%)	TPR (%)	FAR (%)	t_tr (flow/s)	t_pr (flow/s)	CPU (%)	MEM (MB)
Logistic Regression	99.949	99.989	0.2374	23474	2092050	167	489
kNN (k=5)	99.958	99.973	0.1305	9615	2193	100	767
RF (n=100,m=10)	99.992	99.996	0.0297	11764705	109890	1022	717
SVM (C=1,kernel='lin')	99.977	99.995	0.1246	44119	166749	109	547

Table 4: Performance output of DL classifiers with standard library (keras-tensorflow) on "dataset1"

Model	ACC (%)	TPR (%)	FAR (%)	t_tr (flow/s)	t_pr (flow/s)	CPU (%)	MEM (MB)	GPU MEM (MB)
MLP	99.977	99.995	0.1305	710.6	92252	177.7	860	2x 305
1D CNN	99.662	99.901	1.7209	391	13784	886	1447	2x 305
2D CNN	99.705	99.883	1.3233	424	15216	678	1700	2x 305
LSTM	99.958	99.984	0.1899	44.11	7557	767	1331	2x 305
CNN+LSTM	99.952	99.882	0.2196	299	5081	789	1722	2x 305

Table 5: Performance output of ML classifiers with accelerated library (daal4py) on "dataset1"

Model	ACC (%)	TPR (%)	FAR (%)	t_tr (flow/s)	t_pr (flow/s)	CPU (%)	MEM (MB)
Logistic Regression	99.674	99.820	1.2106	31418	11723891	687	413
kNN (k=5)	99.987	99.991	0.0712	801752	9593	773	554
RF (n=100,m=10)	99.999	99.997	0.0238	362780	2234878	1073	509
SVM (C=1,kernel='lin')	99.993	99.997	0.1365	0.1165	1428268	118	581

Table 6: Performance output of DL classifiers with accelerated library (openvino) on "dataset1"

Model	ACC (%)	t_pr (f/s)	CPU (%)	MEM (MB)
MLP	99.977	165304	37.5	909
1D CNN	99.662	31359	73.6	1676
2D CNN	99.705	56635	67.4	1687
LSTM	99.958	28477	46.5	1267
CNN+LSTM	99.952	14495	88.2	1427

standard kNN where a single core is utilized, accelerated kNN is able to allocate the process onto 7.73 cores on average for parallel processing which contributes to the acceleration. Moreover, we observe that accelerated random forest classifier achieves the best accuracy, detection rate, and false alarm rate with more than 20 times acceleration in the prediction stage. Similar to standard counterpart, accelerated RF classifier uses 10.7 cores on average and requires around 30% less memory. Likewise, accelerated SVM model achieves a higher accuracy and detection rate with a lower false alarm rate when compared to the standard library SVM classifier. Even though the training stage for the accelerated tool is much slower than standard SVM, it can classify up to 8.7 times more flows in prediction stage. Since the training stage is executed once to obtain the model, the prediction stage speed is more important throughout the lifetime of the detection system.

5.4 Performance results with OpenVINO-accelerated DL

The deep learning models trained using tensorflow in the previous section are saved as pre-trained models and converted to intermediate representations for OpenVINO to accelerate the prediction stage. Table 6 presents the results obtained using OpenVino toolkit for acceleration. Since MLP model is not supported by DAAL, we first trained MLP model using keras-tensorflow, similar to other DL models, and saved the trained model for OpenVino inference.

One major advantage of accelerated DL models is that although similar amount of memory is utilized, they require a single core for the prediction stage and less amount of memory. For example, MLP model requires 0.37 cores on average, while the most complicated model which is hybrid CNN+LSTM is utilized by 0.88 cores on average during prediction. Moreover, the performance of the DL models on the prediction stage are significantly accelerated. To this end, LSTM model is accelerated the most by 3.77 times and MLP model is accelerated the least by 1.47 times.

Finally, we compare the proposed models with their accelerated counterparts for "dataset2". Figure 4 provides the comparison results. The most accurate model is SVM model with the highest acceleration ratio. Logistic regression has the highest throughput in the prediction stage; however, it is the second least accurate model right before 1D CNN model. Moreover, the accuracy obtained from CNN+LSTM and LSTM models are slightly lower than SVM yet their throughput in the prediction stage significantly lower than SVM. Comparing these results to dataset1 shows that there is not a single model that achieves the highest accuracy.

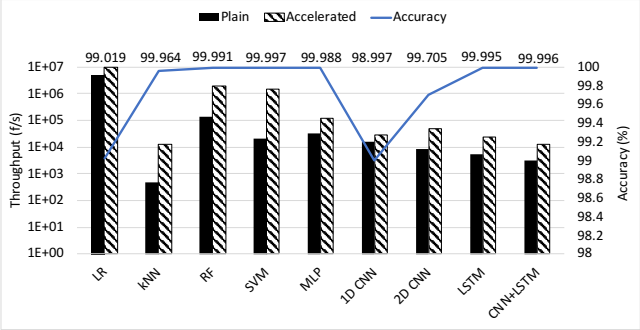

Figure 4: Performance comparison of accelerated classifiers on "dataset2"

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented a comprehensive performance study on machine learning and deep learning based methods to detect encrypted malware. Our results show that SVM, RF and MLP models are the most accurate classifiers. However, there is a trade-off in the model selection. For example, SVM model for CICIDS2017 dataset is the most accurate but RF model achieves very similar accurate yet a higher throughput in the prediction. Similarly, for dataset1, RF model has the highest detection rate with the lowest false alarm rate while SVM model has comparable detection rate with worse false alarm rate. However, SVM is almost 1.5 times faster in prediction speed. Moreover, we observe that with the flow feature extraction tool and careful feature selection, traditional machine learning models may achieve higher accuracy than deep learning models. In the future, we plan to use raw-bytes of header fields in the encrypted flows to train deep learning model to evaluate the effect of flow features. We also plan to study the resource utilization of other stages of malware detection including feature extraction.

ACKNOWLEDGMENTS

This work was supported in part by a grant from Intel Corporation, and a grant from Summer Scholarship, Creative Arts and Research Projects (SCARP) Program of Elizabethtown College.

REFERENCES

[1] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim. 2018. Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection. *IEEE Access* 6 (2018), 33789–33795. https://doi.org/10.1109/ACCESS.2018.2841987

[2] R. Alshammari and A. N. Zincir-Heywood. 2008. Investigating Two Different Approaches for Encrypted Traffic Classification. In *2008 Sixth Annual Conference on Privacy, Security and Trust*. 156–166.

[3] Blake Anderson, Subharthi Paul, and David McGrew. 2016. Deciphering Malware's use of TLS (without Decryption). *Journal of Computer Virology and Hacking Techniques* (07 2016). https://doi.org/10.1007/s11416-017-0306-6

[4] Karel Bartos and Michal Sofka. 2015. Robust Representation for Domain Adaptation in Network Security. In *Machine Learning and Knowledge Discovery in Databases*, Albert Bifet, Michael May, Bianca Zadrozny, Ricard Gavalda, Dino Pedreschi, Francesco Bonchi, Jaime Cardoso, and Myra Spiliopoulou (Eds.). Springer International Publishing, Cham, 116–132.

[5] Canadian Institute for Cybersecurity. 2004. ISCX Datasets. https://www.unb.ca/cic/datasets/index.html [Online; accessed 23-July-2020].

[6] CICIDS2017 General Information. 2020. https://www.unb.ca/cic/datasets/ids-2017.html [Online; accessed 22-July-2020].

[7] Blake Anderson David McGrew et al. 2017. Joy. https://github.com/cisco/joy

[8] Dewan Farid, Harbi Nouria, and Mohammad Zahidur Rahman. 2010. Combining Naive Bayes and Decision Tree for Adaptive Intrusion Detection. *International Journal of Network Security & Its Applications* 2 (04 2010). https://doi.org/10.5121/ijnsa.2010.2202

[9] Minghui Gao, Li Ma, Heng Liu, Zhijun Zhang, Zhiyan Ning, and Jian Xu. 2020. Malicious Network Traffic Detection Based on Deep Neural Networks and Association Analysis. *Sensors* 20 (03 2020), 1452. https://doi.org/10.3390/s20051452

[10] Google. 2020. Encrypted traffic across Google. https://transparencyreport.google.com/https/overview?hl=en [Online; accessed 20-June-2020].

[11] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. 253–262. https://doi.org/10.5220/0006105602530262

[12] S. Haider, A. Akhunzada, I. Mustafa, T. B. Patel, A. Fernandez, K. R. Choo, and J. Iqbal. 2020. A Deep CNN Ensemble Framework for Efficient DDoS Attack Detection in Software Defined Networks. *IEEE Access* 8 (2020), 53972–53983.

[13] W. Hu, W. Hu, and S. Maybank. 2008. AdaBoost-Based Algorithm for Network Intrusion Detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38, 2 (April 2008), 577–583.

[14] Intel Corporation. 2019. Intel® Data Analytics Acceleration Library (Intel® DAAL). https://software.intel.com/en-us/intel-daal

[15] Intel Corporation. 2019. OpenVINO Toolkit: Develop Multiplatform Computer Vision Solutions. Explore the Intel® Distribution of OpenVINO™ toolkit. https://software.intel.com/en-us/openvino-toolkit

[16] KDD Cup 1999 Data. 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [Online; accessed 14-March-2020].

[17] J. Kim, J. Kim, H. L. Thi Thu, and H. Kim. 2016. Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. In *2016 International Conference on Platform Technology and Service (PlatCon)*. 1–5.

[18] Y. Liu, S. Liu, and X. Zhao. 2018. Intrusion Detection Algorithm Based on Convolutional Neural Network. *DEStech Transactions on Engineering and Technology Research* (03 2018). https://doi.org/10.12783/dtetr/iceta2017/19916

[19] Y. Luo, K. Xiang, and S. Li. 2008. Acceleration of Decision Tree Searching for IP Traffic Classification. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '08)*. New York, NY, USA, 40–49. https://doi.org/10.1145/1477942.1477949

[20] D. Manning, P. Li, X. Wu, Y. Luo, T. Zhang, and W. Li. 2020. ACETA: Accelerating Encrypted Traffic Analytics on Network Edge. In *IEEE ICC 2020*. 1–6. https://doi.org/10.1109/ICC40277.2020.9148798

[21] G. Marín, P. Casas, and G. Capdehourat. 2019. Deep in the Dark - Deep Learning-Based Malware Traffic Detection Without Expert Knowledge. In *2019 IEEE Security and Privacy Workshops (SPW)*. 36–42.

[22] M. Panda, A. Abraham, and M. R. Patra. 2010. Discriminative multinomial Naïve Bayes for network intrusion detection. In *2010 Sixth International Conference on Information Assurance and Security*. 5–10. https://doi.org/10.1109/ISIAS.2010.5604193

[23] P. Prasse, L. Machlica, T. Pevný, J. Havelka, and T. Scheffer. 2017. Malware Detection by Analysing Network Traffic with Neural Networks. In *2017 IEEE Security and Privacy Workshops (SPW)*. 205–210.

[24] Vinayakumar R, Soman Kp, and Prabaharan Poornachandran. 2017. Evaluating effectiveness of shallow and deep networks to intrusion detection system. 1282–1289. https://doi.org/10.1109/ICACCI.2017.8126018

[25] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. 108–116. https://doi.org/10.5220/0006639801080116

[26] Stratosphere. 2015. Stratosphere Laboratory Datasets. https://www.stratosphereips.org/datasets-overview [Online; accessed 12-March-2020].

[27] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani. [n.d.]. A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium. Computational Intelligence for Security and Defense Applications* ([n. d.]). https://doi.org/10.1109/CISDA.2009.5356528

[28] Sumaiya Thaseen and Aswani Kumar Cherukuri. 2016. *Intrusion Detection Model Using Chi Square Feature Selection and Modified Naïve Bayes Classifier*. Vol. 49. 81–91. https://doi.org/10.1007/978-3-319-30348-2_7

[29] J. Wang, Q. Yang, and D. Ren. 2009. An Intrusion Detection Algorithm Based on Decision Tree Technology. In *2009 Asia-Pacific Conference on Information Processing*, Vol. 2. 333–335. https://doi.org/10.1109/APCIP.2009.218

[30] P. Wang, F. Ye, X. Chen, and Y. Qian. 2018. Datanet: Deep Learning Based Encrypted Network Traffic Classification in SDN Home Gateway. *IEEE Access* 6 (2018), 55380–55391. https://doi.org/10.1109/ACCESS.2018.2872430

[31] B. Yamansavascilar, M. A. Guvensan, A. G. Yavuz, and M. E. Karsligil. 2017. Application identification via network traffic classification. In *2017 International Conference on Computing, Networking and Communications (ICNC)*. 843–848. https://doi.org/10.1109/ICCNC.2017.7876241

[32] M. Yeo, Y. Koo, Y. Yoon, T. Hwang, J. Ryu, J. Song, and C. Park. 2018. Flow-based malware detection using convolutional neural network. In *2018 International Conference on Information Networking (ICOIN)*. 910–913.

[33] C. Yin, Y. Zhu, J. Fei, and X. He. 2017. A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. *IEEE Access* 5 (2017), 21954–21961.

[34] J. Zhang, M. Zulkernine, and A. Haque. 2008. Random-Forests-Based Network Intrusion Detection Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, 5 (Sep. 2008), 649–659. https://doi.org/10.1109/TSMCC.2008.923876