

Light-weight Unsupervised Anomaly Detection for Encrypted Malware Traffic

Shangbin Han

School of Cyber Science and Technology
Beihang University
Beijing, China
Email: hanshangbin@buaa.edu.cn

Han Zhang

INSC&BNRist
Tsinghua University
Beijing, China
Email: zhhan@tsinghua.edu.cn

Qianhong Wu

School of Cyber Science and Technology
Beihang University
Beijing, China
Email: qianhong.wu@buaa.edu.cn

Bo Qin

School of Information
Renmin University of China
Beijing, China
Email: bo.qin@ruc.edu.cn

Abstract—Users and businesses in the network frequently suffer from attacks by malware like privacy breach. While encrypted traffic protects users and businesses, it also provides convenience for attackers to avoid detection. Existing anomaly detection systems use supervised learning with high-dimension features and employ experts for labeling. However, our exploration reveals that high-dimension features will reduce the efficiency of the classification model. Besides, their training needs abundant high-quality labels, which is difficult to obtain in practice. Facing these challenges, in this paper, we propose an unsupervised anomaly detection method, which adopts the three-layer Autoencoder for feature compression to improve model running efficiency and employs the classical Kmeans algorithm to achieve unsupervised classification. When training the Autoencoder, we only use the normal encrypted traffic. We compare the performance of our method against the state-of-the-art anomaly detection algorithms using open encrypted malware traffic data set. The results demonstrate that our method can achieve the F1-measure of 0.95, which is competitive with supervised learning algorithms.

I. INTRODUCTION

In recent years, the rapid development of the Internet has continuously changed people's work and lifestyle. With the increasing demand for data security and privacy protection, numerous services and applications are using encryption as the primary method of securing information. As of Mar. 6, 2022, 95% of all Google web traffic is encrypted [1]. What is certain is that the encryption trend will continue and may further accelerate in the future.

Encryption technology provides greater privacy and security for businesses online. However, users and businesses are not the ones enjoying the data privacy and protection that encryption brings. Malicious attackers are also using these benefits to protect their malicious behavior. Therefore, the randomness of traffic data makes content inspection extremely difficult. WatchGuard Technologies revealed that 91.5% of all malware threats were delivered by being encrypted [2]. In 2021, malware costs an estimated \$500 billion per month, and

the total cost will reach \$10.5 trillion per year by 2025 [3]. Therefore, mining malicious behavior in the encrypted traffic is especially important.

A rich body of literature exist on detecting malware from encrypted traffic [4], [5], [6], [7], [8], [9]. However, we find that they have the following limitations. Firstly, features are redundant. The feature dimension is high, and there are correlations between many features. One problem is that it slows down the model running. It may also affect the detection performance of the model. Secondly, they have heavy reliance on labels. Unfortunately, it is difficult to obtain abundant high-quality labels in practice. Labeling requires substantial expertise and a lot of manual efforts, and often takes a long time. Besides, malicious attacks have an evolutionary nature, which makes it impossible for labels to keep up with trends in time. Therefore, in most cases, only a few data are labeled. And the quality of labels is not guaranteed. To overcome the above challenges, in this paper, we make the following contributions.

Firstly, we find that feature redundancy (high dimension) affects the efficiency of machine learning algorithms, and the label noise affects the model performance.

Secondly, we propose a three-layer Autoencoder for feature compression to improve efficiency, which is light-weight and has low resource requirements.

Thirdly, we advocate an unsupervised anomaly detection algorithm, and this algorithm can work when there are no labels at all.

Fourthly, we compare the performance of our method against the state-of-the-art anomaly detection algorithms (e.g., SVM [10], LSTM [11], Kitsune [12]) using the open encrypted malware traffic data set [13]. The results show that our method can achieve the F1-measure of 0.95, which is competitive with supervised learning algorithms.

The remainder of the paper is organized as follows. In section II, we introduce the background and motivation. Section

III presents the design details of our method. We evaluate the performance of our method in section IV. Section V shows the related works. Section VI concludes this paper.

II. BACKGROUND AND MOTIVATION

Transport Layer Security (TLS) is a protocol built on top of the transport layer TCP protocol and serves the application layer. It realizes the function of encrypting the packets of the application layer and then sending to TCP for transmission. TLS implements three network security functions: confidentiality, integrity and authentication.

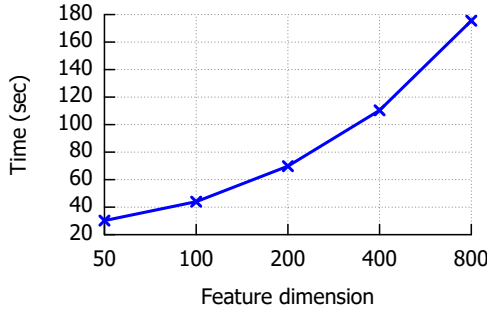


Fig. 1: The training time of SVM grows as the feature dimension increases.

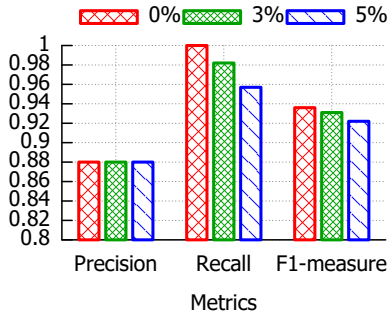


Fig. 2: The performance of SVM degrades with more noisy labels. $x\%$ is the percentage of noisy labels.

The use of encryption technology makes the detection of abnormal traffic difficult. Fortunately, some meta data are visible when encrypting, e.g., the TLS handshake and the certificate information. Using these data, we can extract relevant data features. However, the dimension of the processed features is very large, which may be a challenge for the machine learning algorithm. We use the SVM as an example to illustrate this problem. As shown in Fig. 1, the training time of SVM increases exponentially with the increase of feature dimension. High feature dimension will obviously reduce the efficiency of the algorithm.

Besides, the supervised learning algorithms dominate, whether traditional machine learning or deep learning methods. The key to such schemes is the need for labels with

high-quality ground truth. To this end, the labels are often constructed by domain experts with specialized knowledge, which is highly labor-intensive and expensive [14]. Also, there is no guarantee that the labels are 100% correct. Noise in the labels can affect the performance of the classifier. As shown in Fig. 2, we plot the performance of the SVM algorithm under different scales of noisy labels. $x\%$ means the percentage of noisy labels. We can see that 5% noise causes a 4% performance drop. In practice, the proportion of noisy labels is likely to be larger.

III. METHODOLOGY

In this section, we present the unsupervised anomaly detection framework for encrypted malware traffic. The framework consists of four components: the traffic collector, the feature extraction, the feature compression and the anomaly detection algorithm. Traffic collection component is used to capture network traffic from network devices. NetFlow [15] and IPFIX (IP Flow Information Export) [16] are widely adopted since they monitor rich information. The output of traffic collection is a amount of flow traces (e.g., PCAP) which can be used for network security analysis. The details of traffic collection is omitted here, which is not our focus in this work.

A. Feature extraction

Feature extraction is the process of converting real-world data into numerical vectors. For each flow that has the same five-tuple (source IP address, destination IP address, source port, destination port and protocol), we compute a set of features from the captured encrypted network traffic to provide the best classification accuracy.

TABLE I: Summary of the statistics. e_i can be a set of observed packet sizes.

Type	Name	Notation	Calculation
1D	Count	n	n
1D	Sum	s	$\sum_{i=1}^n e_i$
1D	Min.	mi	$\min\{e_1, \dots, e_n\}$
1D	Max.	ma	$\max\{e_1, \dots, e_n\}$
1D	Mean	μ	s/n
1D	Std.	σ	$\sqrt{\frac{1}{n} \sum_{i=1}^n (e_i - \mu)^2}$
1D-array	Arr.	p_k	$\sum_{k=1}^r e_k$
2D	Mat.	$p_{k,l}$	$\sum_{k=1}^r \sum_{l=1}^t e_{k,l}$
1D	Text	str	str

TABLE I shows the list of the statistics which can be extracted from the traffic packet traces. We refer to the computing statistics involving one and two statistics as one-dimensional statistics and two-dimensional statistics, respectively. Specifically, we extract the features from the traffic

TABLE II: The features extracted from traffic packet trace.

Statistics	Extracted from	# Features	Description
n	Packet, inter-packet time	2	Total, forward and backward traffic, respectively
s, mi, ma, μ, σ	Inter-packet time, TCP window size	10	Total, forward and backward traffic, respectively
s, mi, ma	Packet's byte	3	Total, forward and backward traffic, respectively
μ, σ	Packet's byte per packet/second	4	Total, forward and backward traffic, respectively
s	TCP flags	8	Total, forward and backward traffic, respectively
p_k	Packet's byte distribution	256	Probability of occurrences of each byte value of a flow
$p_{k,l}$	Adjacent packet's byte	625	Probabilistic transition matrix of byte value
μ, σ	Signature key size	2	TLS Certificate
str	Certificate issuer	1	TLS Certificate
str	Cipher suite and extensions	2	TLS handshake
str	Signature algorithm, server name	2	TLS handshake

between the packets' sender and receiver. On the basis of the work presented by Anderson et al. [4], we extend the feature set. The statistics summarize all of the traffic:

- packet's size between client and server.
- inter-packet time between client and server.
- TCP window size and flags between client and server.
- packet's byte distribution and probabilistic transition matrix.
- TLS certificate.
- TLS handshake metadata.

The list of the extracted features are presented in TABLE II. The statistics s, mi, ma, μ and σ are easy to understand. For the TLS features, we vectorize the text string with one-hot encoding. We mainly elaborate on the generation of p_k and $p_{k,l}$. For the specific feature p_k , it is the probability of occurrences of each byte value in the Data fields of a flow, with the i -th element of the array corresponding to the byte value i . For example, the byte distribution is a set $\{30, 50, 70, 90\}$. It means the occurrence of byte 0 is 30, whose probability is $30/(30+50+70+90)=0.125$. Similarly, the occurrence of byte 3 is 90. Therefore, we get the feature of p_k array $\{0.125, 0.208, 0.292, 0.375\}$. Here, we count the probability of 256 bytes. $p_{k,l}$ is the probabilistic transition matrix of byte value. Assuming MTU is 1500 bytes, we divide it by 60 to map to 25 status. For instance, the extracted byte sequence is $\{54, 72, 98, 102\}$. First, we divide it by 60 to get the status $\{0, 1, 1, 1\}$. Second, we count the number of adjacent status to form a 2D matrix. $0 \rightarrow 1$ appear once, $1 \rightarrow 1$ appear twice. Thus, we get matrix $\{\{0,1\}, \{0,2\}\}$. Finally, calculate the probability to get the matrix $\{\{0,0.333\}, \{0,0.667\}\}$.

B. Feature compression

After getting the features, we will find that the feature set has high dimension and redundancy. On the one hand, it may affect the classification accuracy. On the other hand, it can affect the classification efficiency. To tackle this problem, we adopt the Autoencoder [17] for feature compression. To begin

with, we provide a brief introduction to the structure and function of Autoencoder.

1) *Autoencoder*: Autoencoder is a special kind of artificial neural network, which consists of layers with multiple neurons. As shown in Fig. 3, the Autoencoder is made up of two major modules: the encoder and decoder. The input feature vector $\mathbf{x}$ is mapped to a feature space $\mathbf{z}$ through the encoder E . Then, the feature space $\mathbf{z}$ is mapped back to the original space through the decoder D to obtain the reconstructed sample $\hat{\mathbf{x}}$.

Specifically, there are lines connecting the neurons between each adjacent layer. Each line represents a weight, and a weight matrix W is formed between layers. As shown by the neurons inside the red dashed box in Fig. 3, we first get the dot product of the input and the weight. After that, there is a bias vector, which is added with the dot product result, and the sum is passed to a activation function. The activation function decides whether to send this data to the next neuron. Assuming the output is y , the input is $\mathbf{x}$, and the activation function is f . Then each neuron is calculate as follows: $y = f(\sum_{i=1}^n w_i x_i + b)$.

In addition, the hidden layer is the layer between the input layer and the output layer. An Autoencoder can have multiple hidden layers. Starting from the input, activate one layer, pass to the next layer and activate, until the last layer is activated. The whole process is called the forward propagation. If the entire forward propagation function is h , the overall parameters are the weight W , and the bias vector is b , then it can be expressed as: $\hat{\mathbf{x}} = h_{W,b}(\mathbf{x})$.

2) *Training of Autoencoder*: Training is the process of optimizing model parameters, which requires labels to be adjusted. The label of the Autoencoder is the input data itself, so the Autoencoder can also be regarded as a data generator.

The goal of Autoencoder training is to learn a function h and a set of parameters: weight W and bias vector b , which can satisfy $h_{W,b}(\mathbf{x}) \approx \mathbf{x}$. At the beginning of the model, the parameters of the Autoencoder will be far from ideal. During the entire training process, the data flow is bidirectional. The data flow reaches the output layer through

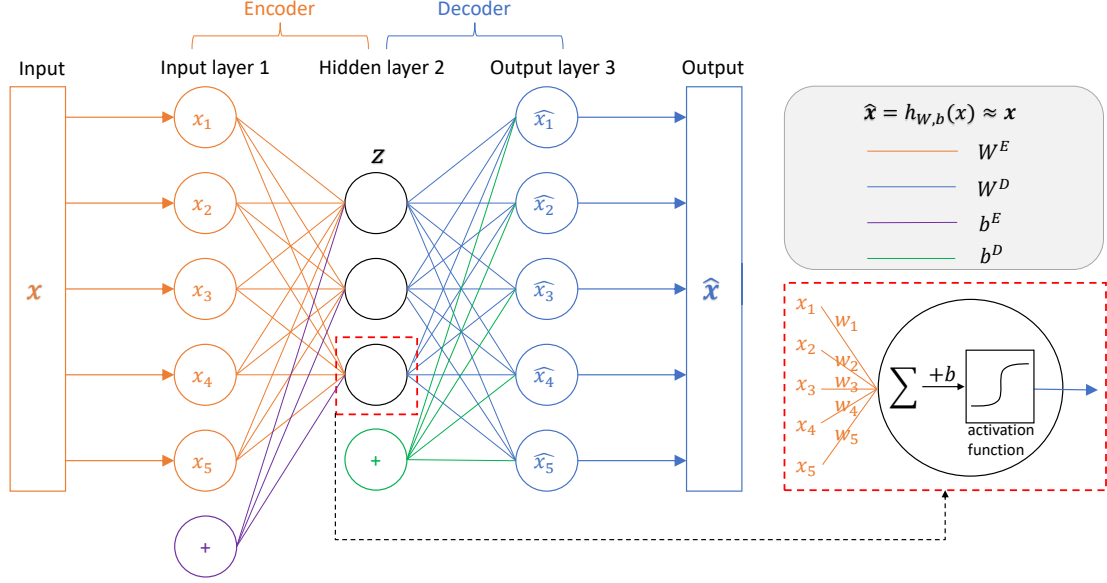


Fig. 3: A simple example of Autoencoder.

forward propagation, and then the parameters of the neurons are recalculated through back propagation.

In back propagation, a loss function needs to be defined, which takes into account both the output of the model and the actual results (labels). The loss function is an indicator of how good the model is. Autoencoder usually use the mean squared error (MSE) to represent its loss function.

$$l(W, b) = \frac{1}{n} \sum_{i=1}^n (h_{W,b}(x) - x)^2 \quad (1)$$

Since the loss function measures the difference between the model predicted value and the actual observed value. Therefore, we want the loss function to be as small as possible, which means that the model output is approximately equal to the actual observed value.

To adjust the parameters, a well-known optimization algorithm is often adopted, that is, gradient descent (GD) [18]. By setting a step size (learning rate), the values of weights and bias are gradually adjusted along the direction of the fastest gradient descent of the loss function.

This process is repeated until the goal is met. When the model is trained to the ideal situation, the gradient should approach 0, and the loss function value will also approach 0. It means that the generated data differs very little from the original data.

3) *Feature compression with Autoencoder*: Autoencoder has been widely used for various machine learning tasks. For instance, performing image compression [19], and generating images [20]. In this work, we focus on the feature compression with Autoencoders.

The Autoencoder first encodes the input feature value x into an invisible feature space z . The decoder then reconstructs

z as $\hat{x}$ with the same distribution as x . If the input and x have different distributions, then the reconstructed data will have a large gap with x . A step forward, their corresponding feature space z also have significant differences. In addition, the dimension of the feature space z is significantly smaller than the input feature vector x . Using z as the new feature vector achieves the purpose of feature compression.

Specifically, we use a three-layer Autoencoder to avoid computational overhead during training. We only input the normal traffic into the Autoencoder as training data. When executing the Autoencoder, only the encoder part is taken to compress the arriving traffic to obtain the compressed feature vector.

C. Anomaly detection

After feature compression, the anomaly detection follows. Existing schemes mainly focus on supervised anomaly detection schemes. They require labels that truly reflect the actual situation. In the field of network anomaly detection, it is impossible to obtain real-time labels with abundant high-quality ground-truth. Therefore, we turn to unsupervised machine learning algorithm. We select clustering algorithm Kmeans for anomaly detection.

IV. EVALUATION

In this section, we evaluate the performance of our method with open encrypted malware traffic data sets. Main results of our experiments can be concluded as:

- We find that high-dimension features affect efficiency, high-quality labels are difficult to obtain and 5% label noise causes a 4% performance drop.

- We adopt Autoencoder to compress the high-dimension features to low-dimension features without loss of accuracy. We use the unsupervised learning algorithm to remove the dependence on expensive labels.
- Compared with state-of-the-art anomaly detection algorithms, our method can achieve the highest F1-measure of 0.95, which is competitive with supervised learning algorithms.

A. Data sets

In our evaluation, we use the open DataCon2020-Encrypted Malicious Traffic Data Set [13] to examine the performance of proposed method. The data set is derived from the malware and normal software collected from February to June 2020. It is run and generated by the Qianxin Technology Research Institute's Sky Vault sandbox. The black samples in the training set are the malware-encrypted traffic captured from February 2020 to May 2020, and the black samples in the test set are the malware-encrypted traffic captured in June 2020. All white samples are normal software encrypted traffic captured in 2020. The malicious traffic defined in this data set is encrypted traffic generated by malicious software, and white traffic is encrypted traffic generated by normal software. The training set has 6000 samples, and the ratio of black and white samples is 1:1. The test size includes 2000 black samples and 2000 white samples.

B. Experiment setup

We use implementations from Keras [21] for Autoencoder and from scikit-learn [22] for Kmeans. The Autoencoder we implemented has three layers, and the number of neurons in the hidden layer is $\frac{1}{4}$ of the input feature dimension. All our experiments are performed on a Linux server, which is configured with 32G memory, 9T hard disk and 16 Intel Xeon E5-2630 CPUs.

C. Baselines

We compare the performance of our method with five methods: one classical unsupervised learning methods density peaks clustering (DPC) [23], two classical supervised learning methods linear SVM [10] and XGBoost [10], two neural network method long short term memory (LSTM, supervised learning) [11] and Kitsune (unsupervised learning) [12].

We use the following metrics to evaluate the performance of Autoperman and baseline methods:

- 1) Accuracy: It shows the correct prediction rate across the whole data set, which is defined as follows: $\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$, where TP is True Positives, TN is True Negatives, FP is False Positives and FN is False Negatives.
- 2) Precision: It calculates the proportion of predicted positive instance that are actually positive instance. It is expressed as: $\text{Precision} = \frac{TP}{TP+FP}$.
- 3) Recall: It represents the ability of a detection model to identify positive instance, which is defined as: $\text{Recall} = \frac{TP}{TP+FN}$.

- 4) F1-measure: It considers both Precision and Recall, and is defined as the harmonic average of the two metric: $\text{F1-measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$.

D. Training of Autoencoder

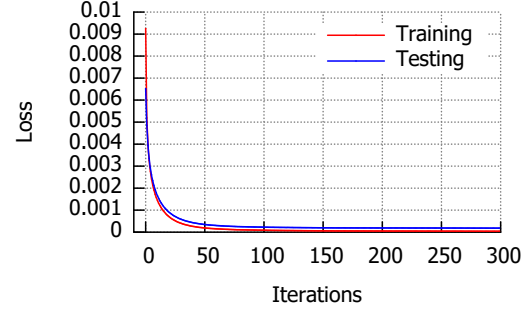


Fig. 4: Loss of Autoencoder.

After training, the loss of an ideal Autoencoder should be close to 0. This ensures that the reconstructed data is approximately equal to the original data. This ideal model determines the network structure of the model as well as the ideal model parameters. As a result, the intermediate invisible vector $\mathbf{z}$ obtained from the ideal model is more descriptive of the original data.

We only collect the normal encrypted malware traffic as the training data. When training the Autoencoder for feature compression, we select *mean squared error* (MSE, eq. (1)) metric as the loss function, and use the *adam* [24] algorithm as the optimizer. A common activation function, and what we adopt in this work, is the *tanh* function, defined as

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (2)$$

Fig. 4 shows the change of the loss with the number of iterations when training the Autoencoder for feature compression. In the first 100 iterations, the loss drops rapidly and converges. We can see that the loss convergence value is almost 0. It means that the model achieves the desired effect.

E. Efficiency after feature compression

As shown in Fig. 1 in Sec. II, we take SVM as an example to present the inefficiency of high-dimensional feature vectors. For comparison, we still use SVM as an example to show the cost of SVM with features after compression. In the experiment, the feature dimension is compressed to $\frac{1}{4}$ of the original data dimension.

Fig. 5 shows the time consuming of SVM in full and compressed dimensions, respectively. Specifically, we characterize their cost under different sample sizes. We can see that the growth rate of time consuming under compressed features is significantly lower than that of full features. As the number of samples increases, the time savings becomes more pronounced. The reason is that dimension compression reduces the amount of data that can be processed.

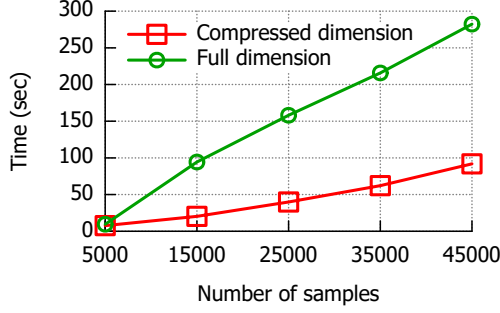


Fig. 5: Cost of SVM.

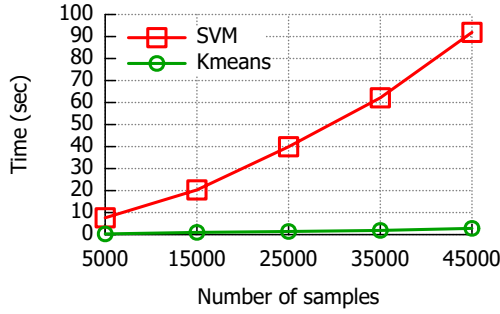


Fig. 6: Cost comparison under compressed features.

Meanwhile, we compare the time consumption of SVM and Kmeans under compressed features, since in our method, Kmeans is adopted as an unsupervised classifier. As shown in Fig. 6, the time consumption of SVM is obviously much more than that of Kmeans. This is because SVM loops over the training data to reach convergence.

F. Compression ratio

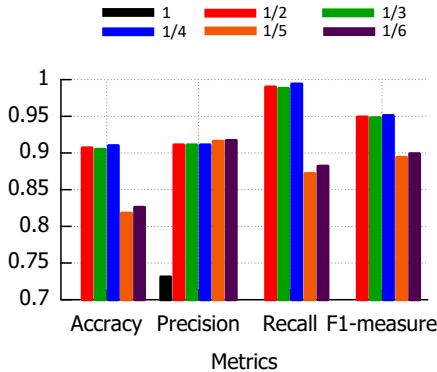


Fig. 7: Performance of executing Kmeans with different compression ratios.

Since the feature dimensions needs to be compressed, what is the appropriate compression ratio (the ratio between the feature compression dimension and the original feature dimension)? In this regard, we carried out experiments, and Fig. 7

shows the results. Except for different values of compression ratio, all other parameters are set with default parameters. Specifically, the compression ratios are 1, 1/2, 1/3, 1/4, 1/5 and 1/6.

It can be seen that when the ratio is 1 (there is no compression), the accuracy, recall and F1-measure are not drawn, because they are all lower than 0.2. This shows that the direct use of original features for unsupervised classification has poor performance. The reason is that the high-dimensional features have strong redundancy and poor differentiation of data distribution. When the compression ratio is 1/2, 1/3, and 1/4, the performance is significantly improved. However, when the compression ratio is 1/5 and 1/6, the performance begins to decline again. The reason is that the Autoencoder is lossy compression. When the dimension is relatively low, the detailed information will be lost.

Therefore, the proper value of compression ratio should be selected. Too high or too low will affect the detection performance of the model. In practice, the compression ratio can be obtained by cross validation.

G. Performance comparison

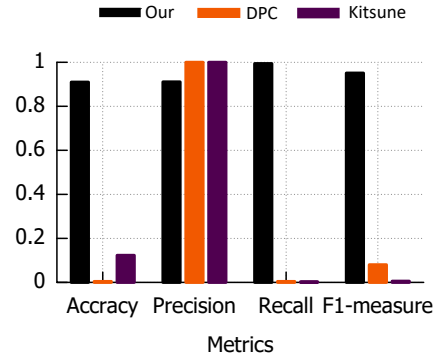


Fig. 8: Unsupervised learning results comparison.

In this part, we compared the performance of our method to the state-of-the-art algorithms.

Fig. 8 shows the experimental results for the state-of-the-art unsupervised learning algorithms along with our method for anomaly detection in encrypted malware traffic. As this algorithms do not need labels to train and build the models, we used the normal encrypted traffic from the white data in the training set. For testing, we used all the test set.

Overall, we observed that the state-of-the-art algorithms have high precision, but all other metrics are very low in comparison to our method even though they were pre-trained with the ground-truth. It means that they cannot detect malicious encrypted traffic.

Besides, we also compared the performance of our method with other state-of-the-art supervised learning algorithms and the results are given in Fig. 9. Overall, the results presented in Fig. 9 clearly show that our method outperforms SVM, XGBoost and LSTM in terms of accuracy, precision and F1-measure. Specifically, the F1-measure achieved by our method

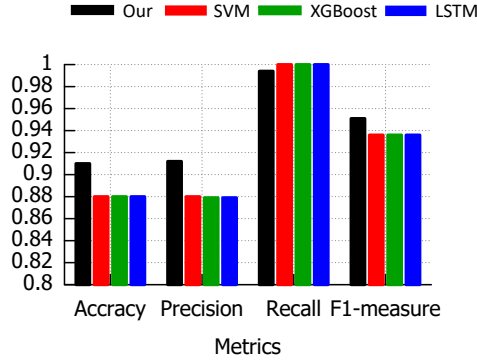


Fig. 9: Supervised learning results comparison.

is highest, reaching 0.95, which is about 5% higher than the supervised learning methods. With regards to accuracy and precision, SVM, XGBoost and LSTM are lower than our method by nearly 3%. Although their recall is slightly higher than our method, they are over-fitting. Considering that our method is unsupervised learning, we conclude that our method does not consume expensive labels and produce high-accuracy models that are competitive with supervised learning algorithms.

V. RELATED WORK

Feature extraction is a key part of realizing various learning tasks, which contains two types of features, protocol-independent features and protocol-dependent features. Protocol-independent features mainly include packet-based and session-based features. Nakahara et al. [25] chose the packet-based features such as packets number, packet length as features to train a detection model. Hafeez et al. [26] extracts session-based features for training a deep learning model. Liu et al. [27] combines packet-based and session-based features for training and traffic identification. Protocol-dependent features are extracted from the encryption protocol. Chen et al. [23] and Shekhawat et al. [10] extract visible meta data from the TLS/SSL protocol. There is also work to merge the two types of features. Anderson et al. [4] proposes an enhanced feature set that combines protocol-independent features and TLS/SSL features. In our work, we also combine them and just make some additions to the feature set.

Supervised learning models. They train anomaly detection classifiers using the labels. Shekhawat et al. [10] considers three machine learning algorithms involving support vector machines (SVM), random forests (RF), and extreme gradient boosting (XGBoost). The results demonstrate that the XGBoost shows the best performance. Ma et al. [5] proposes a model training method based on the K-Nearest Neighbors (KNN) algorithm. Niu et al. [6] proposes a Heuristic Statistical Test (HST) method that combines statistical and machine learning algorithm such as C4.5. Lee et al. [7] proposes an incremental support vector machine method with stochastic gradient descent that maintains an adequate level of perfor-

mance with regular updates. Lee et al. [28] analyzes and proposes several new TLS features to characterize encrypted malware and proposes the use of random forests to identify TLS-encrypted malware traffic. Both schemes show the promising results. However, they heavily rely on the abundant high-quality labels, which is generally not feasible in evolving and large scale network.

Unsupervised learning models. Chen et al. [23] proposes a three-stage stratified sampling method based on an improved density peak clustering algorithm (THS-IDPC) to improve the accuracy and efficiency of encrypted malicious traffic detection models. Liu et al. [27] proposes a distance-based method that utilizes the unsupervised learning algorithm Gaussian Mixture Model (GMM) and Ranked Point Recognition Clustering Structure (OPTICS) to calculate the distance between malware. Hafeez et al. [26] proposes iot-keeper, which combines fuzzy c-means clustering and fuzzy interpolation scheme to detect malicious network activities. Once a malicious behavior is detected, iot-keeper will automatically impose network access restrictions on the IOT devices that generate this activity and prevent them from attacking other devices or services. Fujita et al. [29] proposes a behavior detection model that extracts botnet data from network traffic using K-means clustering algorithm. Gomez et al. [30] proposes a novel unsupervised method to detect and cluster malicious TLS flows. The detector can assign a malicious flow to the cluster it belongs to, or determine if it is benign. The models can be readily trained even without labels.

Deep learning models. Recently, there is a rising trend of employing deep learning algorithms for anomaly detection, e.g., LSTM [11], [31], [32] and Autoencoder [12], [33], [8], CNN [34], [35], RNN [9]. There are also works that combine multiple ANN to build the detection models [36], [37], [38], [39], [40]. Though they have strong learning ability, they have poor interpretability and high resource demand.

VI. CONCLUSION

In this paper, we studied the importance of encrypted traffic and anomaly detection systems. We studied the limitations of existing anomaly detection approaches such as strong feature redundancy, increasing computation overhead, failure to detect new attacks. We then proposed an unsupervised method with feature compression for encrypted malware traffic anomaly detection, which contains traffic collection component, feature extraction component, feature compression component, and unsupervised anomaly detection component. We advocated to adopt Autoencoder to compress the high feature dimension, which solves the efficiency problem brought by high-dimension features. With experiments we demonstrated that our method improves the performance of encrypted malware traffic, and in particular the F1-measure compared to state-of-the-art methods. In the future work, we will explore methods using deep learning and compare with this scheme.

ACKNOWLEDGMENT

This paper is supported by the National Key R&D Program of China through project 2020YFB1005600, the Natural Science Foundation of China through projects U21A20467, 61932011, and 61972019, the Beijing Natural Science Foundation through project M21031, the Populus euphratica found CCF-HuaweiBC2021009.

REFERENCES

- [1] Google, "Google transparency report." Accessed: Mar. 2022. [Online]. Available: <https://transparencyreport.google.com/https/overview?hl=en>, 2022.
- [2] WatchGuard, "Internet security report." Accessed: Mar. 2022. [Online]. Available: <https://www.watchguard.com/wgrd-resource-center/security-report-q2-2021>, 2021.
- [3] A. Sanders, "15 (crazy) malware and virus statistics, trends & facts." Accessed: Mar. 2022. [Online]. Available: <https://www.safetynetdetectives.com/blog/malware-statistics>, 2021.
- [4] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity," in *Proceedings of the 23rd ACM SIGKDD International Conference on knowledge discovery and data mining*, pp. 1723–1732, 2017.
- [5] C. Ma, X. Du, and L. Cao, "Improved knn algorithm for fine-grained classification of encrypted network flow," *Electronics*, vol. 9, no. 2, p. 324, 2020.
- [6] W. Niu, Z. Zhuo, X. Zhang, X. Du, G. Yang, and M. Guizani, "A heuristic statistical testing based approach for encrypted network traffic identification," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3843–3853, 2019.
- [7] I. Lee, H. Roh, and W. Lee, "Encrypted malware traffic detection using incremental learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1348–1349, IEEE, 2020.
- [8] J. Xing and C. Wu, "Detecting anomalies in encrypted traffic via deep dictionary learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 734–739, IEEE, 2020.
- [9] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*, pp. 1171–1179, IEEE, 2019.
- [10] A. S. Shekhawat, F. Di Troia, and M. Stamp, "Feature analysis of encrypted malicious traffic," *Expert Systems with Applications*, vol. 125, pp. 130–141, 2019.
- [11] H. Yao, C. Liu, P. Zhang, S. Wu, C. Jiang, and S. Yu, "Identification of encrypted traffic through attention mechanism based long short term memory," *IEEE Transactions on Big Data*, 2019.
- [12] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.
- [13] Qianxin, "Datacon2020-encrypted malicious traffic dataset." Accessed: Mar. 2022. [Online]. Available: <https://datacon.qianxin.com/opendata/openpage?resourceId=6>, 2020.
- [14] J. Kim, C. Sim, and J. Choi, "Generating labeled flow data from mawilab traces for network intrusion detection," in *Proceedings of the ACM Workshop on Systems and Network Telemetry and Analytics*, pp. 45–48, 2019.
- [15] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "Cisco systems netflow services export version 9," 2004.
- [16] B. Claise, B. Trammell, and P. Aitken, "Specification of the ip flow information export (ipfix) protocol for the exchange of flow information," 2013.
- [17] J. Zhai, S. Zhang, J. Chen, and Q. He, "Autoencoder and its various variants," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 415–419, IEEE, 2018.
- [18] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.
- [19] Y. Choi, M. El-Khamy, and J. Lee, "Variable rate deep image compression with a conditional autoencoder," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3146–3154, 2019.
- [20] W. Xu, S. Keshmiri, and G. Wang, "Adversarially approximated autoencoder for image generation and manipulation," *IEEE Transactions on Multimedia*, vol. 21, no. 9, pp. 2387–2396, 2019.
- [21] F. Chollet, "Keras." Accessed: Mar. 2022. [Online]. Available: <https://github.com/keras-team/keras>, 2015.
- [22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
- [23] L. Chen, S. Gao, B. Liu, Z. Lu, and Z. Jiang, "Ths-idpc: A three-stage hierarchical sampling method based on improved density peaks clustering algorithm for encrypted malicious traffic detection," *The Journal of Supercomputing*, vol. 76, no. 9, pp. 7489–7518, 2020.
- [24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [25] M. Nakahara, N. Okui, Y. Kobayashi, and Y. Miyake, "Machine learning based malware traffic detection on iot devices using summarized packet data," in *IoTBDs*, pp. 78–87, 2020.
- [26] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "Iot-keeper: Detecting malicious iot network activity using online traffic analysis at the edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 45–59, 2020.
- [27] J. Liu, Z. Tian, R. Zheng, and L. Liu, "A distance-based method for building an encrypted malware traffic identification framework," *IEEE Access*, vol. 7, pp. 100014–100028, 2019.
- [28] W. Lee and S. Jin, "Encrypted malware traffic detection using tls features and random forest," in *International Conference on Computational & Experimental Engineering and Sciences*, pp. 85–100, Springer, 2021.
- [29] H. Fujita and A. Selamat, "Clustering botnet behavior using k-means with uncertain data," in *Advancing Technology Industrialization Through Intelligent Software Methodologies, Tools and Techniques: Proceedings of the 18th International Conference on New Trends in Intelligent Software Methodologies, Tools and Techniques (SoMet_19)*, vol. 318, p. 244, IOS Press, 2019.
- [30] G. Gómez, P. Kotzias, M. Dell'Amico, L. Bilge, and J. Caballero, "Unsupervised detection and clustering of malicious tls flows," *arXiv preprint arXiv:2109.03878*, 2021.
- [31] R. Li, Z. Song, W. Xie, C. Zhang, G. Zhong, and X. Pei, "Halnet: A hybrid deep learning model for encrypted c&c malware traffic detection," in *International Conference on Network and System Security*, pp. 326–339, Springer, 2021.
- [32] A. Ferriyan, A. H. Thamrin, K. Takeda, and J. Murai, "Encrypted malicious traffic detection based on word2vec," *Electronics*, vol. 11, no. 5, p. 679, 2022.
- [33] T. Yu, F. Zou, L. Li, and P. Yi, "An encrypted malicious traffic detection system based on neural network," in *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pp. 62–70, IEEE, 2019.
- [34] D. Arivudainambi, V. K. KA, P. Visu, et al., "Malware traffic classification using principal component analysis and artificial neural network for extreme surveillance," *Computer Communications*, vol. 147, pp. 50–57, 2019.
- [35] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International conference on information networking (ICOIN)*, pp. 712–717, IEEE, 2017.
- [36] G. Marín, P. Caasas, and G. Capdehourat, "Deepmal - deep learning models for malware traffic detection and classification," in *Data Science–Analytics and Applications*, pp. 105–112, Springer, 2021.
- [37] X. Hu, C. Gu, and F. Wei, "Cld-net: a network combining cnn and lstm for internet encrypted traffic classification," *Security and Communication Networks*, vol. 2021, 2021.
- [38] S. Liu, Y. Han, Y. Hu, and Q. Tan, "Fa-net: Attention-based fusion network for malware https traffic classification," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–7, IEEE, 2021.
- [39] X. Wang, S. Chen, and J. Su, "Automatic mobile app identification from encrypted traffic with hybrid neural networks," *IEEE Access*, vol. 8, pp. 182065–182077, 2020.
- [40] Y. Zeng, H. Gu, W. Wei, and Y. Guo, "Deep-full-range: a deep learning based network encrypted traffic classification and intrusion detection framework," *IEEE Access*, vol. 7, pp. 45182–45190, 2019.