MalPaCA Feature Engineering - A comparative analysis between automated feature engineering and manual feature engineering on network traffic

Sung kyung Park , Azqa Nadeem , Sicco Verwer
TU Delft
sungpark@student.tudelft.nl, azqa.nadeem@tudelft.nl, s.e.verwer@tudelft.nl

Abstract

Identifying novel malware and their behaviour enables security engineers to prevent and protect users with devices on the network from attackers. MalPaCA is an algorithm that helps to understand the behaviours of the network traffic by clustering uni-directional network connections which can be analyzed further to interpret which label suites the malicious connection. When clustering connections, features extracted from the packet information were chosen manually based on the generalizability of information and research of common malware characteristics. The feature set can be extracted automatically with an autoencoder to increase the representation of each packets in network traffics. A comparison with an autoencoder generated feature set to the hand-crafted feature set shows that the hand-crafted feature set represents the malicious traffics with higher accuracy and more insightful explainability. A comparative experiment is run on the IoT-23 dataset, a network traffic capture from Avast's AIC laboratory.

1 Introduction

In recent years, malware attacks have been increasing, with a survey showing that 70% of devices connected to the Internet are vulnerable [1]. It has come to close attention to keep devices connected to the Internet protected from malware. To do so, new malware need to be identified and the behaviour of novel malware should be understood to make high-tech solutions to defend against them.

Malware attacks are executed through the network, hence the history of the malicious network flows are left on the network traffic which can be inspected to identify and understand the malware. Cyber-security engineers manually analyze the network traffic to make behavioural profiles of malware families on the network which becomes outdated soon after [2]. With MalPaCA (Malware Packet-sequence Clustering and Analysis), a novel unsupervised machine learning-based method which automates capability assessment by clustering the temporal behavior in malware's network traces, provides meaningful behavioral clusters using packet headers [3].

Within the pipeline of the MalPaCA algorithm, feature extraction plays a role in the early stages as shown in Figure 1. The basic features available from the network traffic are the IP and TCP/UDP fields without the payload to avoid deep packet inspection for minimal computational load. The feature extraction in the current MalPaCA algorithm is hand-crafted, meaning a feature set to be used for clustering has been selected through literature study. A well representative feature set that represents the packet, hence a connection, will collect values that will enhance clustering of connections, thus a search for new feature sets has been questioned.

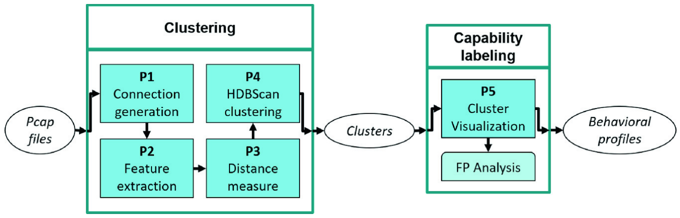


Figure 1: MalPaCA pipeline

The integration of an autoencoder (AE) to extract a new feature set in the form of data encoding representing all the features available from the packet header is the automated feature engineering version of MalPaCA. The key benefit of this approach is that a representative feature set containing all information from a packet header can be generated automatically in any situation, including where the domain of the network traffic is unclear. Such compressed knowledge representation helps to learn important hidden features. The research of an automated feature engineering MalPaCA tries to contribute for a better behavioural profile of the network connection clusters. With the automatically extracted feature set, the existing manually extracted feature set will be compared to evaluate accuracy and explainability of the clusters.

In this paper, the formulation of an autoencoder to extract a new feature set will be discussed. A comparison of the new automatically extracted feature set against the manually extracted feature set will be run with MalPaCA on network traffics from the IoT-23 dataset. The result will be evaluated quantitatively to demonstrate accuracy and qualitatively to explain how the new feature set differ in resulting cluster and the explainability of the clusters which are used to help label the cluster and learn the malware behaviour.

In section 2, the background of MalPaCA will be discussed, focused on the P2 feature extraction. In section 3, the integration of the automated feature engineering to MalPaCA will be described as the AE integrated MalPaCA version. In section 4, the experimental set up including the dataset, parameter settings, and evaluation metrics will be explained. In section 5, the results of the experiment will be interpreted, with an analysis on section 6. Section 7 discusses the ethics and reproducibility of the research. Finally, conclusions and future work in section 7 remarks the end of the research.

2 Background

Malware packet sequence clustering and analysis, namely MalPaCA, is an analysis tool that performs automated capability assessment to construct a behavioral profile for each malware sample that is more descriptive than just its family label [3]. The unsupervised learning manner of MalPaCA puts emphasis to the explainability of cluster results. The algorithm's pipeline is shown in Figure 1 with five main steps from P1 to P5. Uni-directional connections are generated in P1 and in P2 a feature set formed of four features are extracted from each packet in every connections. Each uni-directional connection are formed of a threshold value number of packets and each packet consists four features. In P3, dynamic time warping or n-gram distance measures are used to create a *n*-by-*n* distance matrix where *n* is the number of uni-directional connections. With four *n*-by-*n* distance matrix, an aggregated distance matrix is formed by obtaining the average. This aggregated distance matrix is fed into the HDBSCAN to acquire the final clustering of uni-directional connections in P4. This is the clustering part of the baseline MalPaCA.

Few of the challenges stated in MalPaCA are feature selection and feature representation which are composite aspects of P2 [3]. As for feature selection, deep packet inspection has been avoided for the reasons such as privacy-intrusive, operationally expensive, and do not work out-of-the-box for encrypted traffic characteristics. Hence, MalPaCA selects high-level features from packet headers. In feature representation, MalPaCA employs short raw sequential features instead of statistical features that are widely used in characterizing malware network behavior, such as mean packet size of a network flow [4, 5]. With raw sequential features, behaviours of each packets are fully detailed without any loss in local behaviours. Accompanying these two challenges, MalPaCA chose packet size, time interval, source port and destination port as high-level raw sequential features from packet headers. These four features were hand-crafted with the reason of generalizablility to more than one type of malware and being small and easy to extract [3]. All four features are available for every connection and each feature is represented as a sequence of raw observations for subsequent packets.

3 Methodology

The comparative analysis relies heavily on two different complete versions of the MalPaCA algorithm. One of which is the baseline MalPaCA [3] and the other is MalPaCA with an automated feature extraction generated by an autoencoder replacing the original feature set with encoded values. To do so, a novel version of MalPaCA should be made to incorporate a feature set accurately encoded and continue the algorithm to flow to P3, namely, distance measure. The new version of MalPaCA with the automated feature engineering will be described in this section. Changes within the steps of clustering are modified while keeping the rest of the algorithm controlled to the original MalPaCA.

3.1 Autoencoder for dimensionality reduction

An essential part of this research devotes to, the artificial neural network namely, the autoencoder. The autoencoder is used in the automated feature engineering version of MalPaCA to perform dimensionality reduction on the feature set to accomplish feature extraction before putting into clustering. Autoencoders are capable of modeling complex non-linear functions, hence on inputs such as network traffic, it is idealistic. A dimension reduced encoding of datasets such as network traffic can increase the representativeness of each data instances by denoising the dataset and reducing overfitting of the model to better perform clustering [6].

3.2 MalPaCA: autoencoder integrated

In the AE integrated MalPaCA, P2, feature extraction is replaced. With the autoencoder designed in subsection 4.2, the modified MalPaCA pipeline look as in Figure 2.

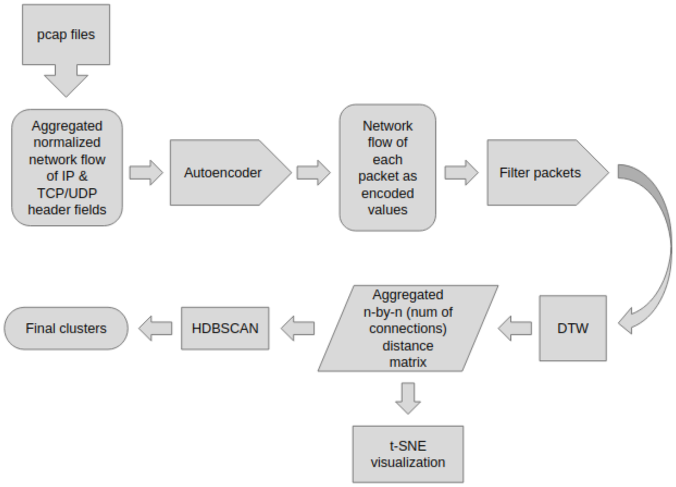

pcap files
Aggregated normalized network flow of IP & TCP/UDP header fields
Autoencoder
Network flow of each packet as encoded values
Filter packets
DTW
Aggregated n-by-n (num of connections) distance matrix
HDBSCAN
Final clusters
t-SNE visualization

Figure 2: Automated feature engineering MalPaCA pipeline

A list of pcap files or a pcap file is put in as input to MalPaCA. All packets in the pcap file(s) are combined into a single list of 12-tuple. The high-level features from packet headers form the 12-tuple for each packet. This list, representing the network traffic, is normalized then put into the autoencoder, returning a list of encoding. An example of this encoding is [0.3772512, 0.14221649, -0.03652108, 0.31472498, 0.33470532] with latent space of five, further elaborated in subsection 4.3. With each packets of the network traffic encoded, the newly generated list is an encoded representation of the network traffic input. This list of encoded values are filtered, reducing the size of the input to only keep the packets used in MalPaCA. The packets that are not used in the algorithm are trimmed out further explained in subsection 4.1. With the encoded feature set for each packet in connections, dynamic time warping is used to measure the similarity between all connections. This creates a n-by-n distance matrix between all uni-directional connections. Finally, HDBSCAN and t-SNE visualization are performed on this distance matrix resulting in a cluster of connections and a visualization of the clusters, respectively.

4 Experimental setup

The experiment of the two configuration of MalPaCA shown in this section are dedicated to exhibit the effectiveness of automated feature engineering and identifying malware behaviours from network traffic. In this section, the experimental setup including the autoencoder design, dataset and metrics will be discussed.

4.1 Experimental dataset

All the network traffic scenarios to be used in this research come from the IoT-23 dataset created by Avast's AIC laboratory. The IoT-23 dataset consists 23 network captures from Internet of Things devices where 20 captures are malicious network traffics and 3 are benign captures [7]. For each capture, a pcap file consists several thousands of packet flows that go up to 7.8GB in size. The IoT-23 dataset is a real network traffic dataset consisting 15 different connections of malware labels created manually in the Stratosphere laboratory considering the malware captures analysis, and benign connections.

One of the advantages of MalPaCA is that the computational load can be reduced, due to the threshold value that limits the number of packets to consider per each uni-directional connections. Uni-directional connections that have packets less than the threshold number are ignored. This threshold value is by default set to 20 in MalPaCA. Taking advantage of this trait of MalPaCA, data preparation can optimize the algorithm by filtering out unnecessary packets from the pcap file. These packets include packets that are from a uni-directional connection that has less than the threshold value or packets that come after the first threshold number of packets from uni-directional connections that have packets more than the threshold number of packets.

All 23 captures of the network traffic from the IoT-23 dataset were partially used by filtering out certain packets not required for MalPaCA. The labels of each connections are given from the IoT-23 dataset made by Zeek network analyzer. Table 1 shows the malware labels of the pre-processed dataset used and the distribution of each labels, after filtering out unused packets (threshold is set to 20 as default). 69.9% of the network traffic are benign and the rest of 30.1% are malicious traffic in the new folder of pcap files.

4.2 Autoencoder model

An autoencoder that will map the feature set from a high feature space to low feature space was carefully designed for MalPaCA. The high feature space are the basic features from IP and TCP/UDP header fields that are available from the network traffic without inspecting the payload. This is a tuple of 12 values which are in the order of packet size, flags, destination IP address, fragment offset, protocol, source IP address, header checksum, type of service, time to live, source port, destination port and TCP/UDP checksum. Hence the input to the autoencoder will be an array of 12-tuple where each tuple is a single packet from a pcap file.

Table 1: Dataset composition filtered from IoT-23 dataset

Label	Num. of connections	Percentage
Benign	956	69.9%
Attack	37	2.7%
C&C	26	1.9%
C&C-heartbeat	13	1.0%
C&C-heartbeat-file-download	4	0.3%
C&C-Torii	40	2.9%
DDoS	62	4.5%
File-download	2	0.1%
Okiru	21	1.5%
Part-of-a-horizontal-port-scan	207	15.1%
Total	1368	100%

An autoencoder has three main components which are the encoder, latent variables and decoder. The encoder and decoder are symmetrical neural networks of which the decoder is discarded after learning how to recreate the input as the output from the latent variables. The latent variable is a layer which is the bottleneck in the neural network representing the high feature space into a low feature space.

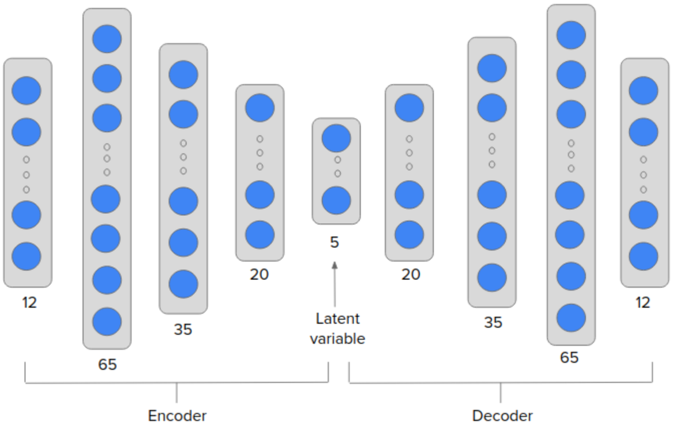


Figure 3: Autoencoder for MalPaCA structure

The autoencoder for MalPaCA has been structured as shown in Figure 3. To design this model that learns compact yet accurate representation of network traffic for MalPaCA, literature study and grid search have been employed. Below are the explanations for each choice of the structure in the autoencoder.

Autoencoder variant

Amongst the variants of an autoencoder, the undercomplete autoencoder will be used. Undercomplete autoencoder constrains the number of nodes in the hidden layers to be lower, learning nonlinear relations, to obtain latent representations of the input which is the objective of for the automated feature engineering MalPaCA.

Number of hidden layers

There will be an input layer, output layer and seven hidden layers in total where the encoder and decoder each have three layers with one latent variable layer in the middle. Two hidden layers can represent functions with any kind of shape sufficiently for many practical problems, but with large input data such as network traffics [8], the use of additional hidden layers reduce the total required number of hidden nodes which help in learning smaller latent variable space [9].

Number of neurons

[12, 65, 35, 20, 5, 20, 35, 65, 12] is the structure of the autoencoder where each value represents the number of neurons in that layer. 12 is the feature space of the input, 65 is the number of neurons in the first and last hidden layer, and five is the size of the autoencoder's bottleneck layer. The five latent variable unit provide a five-dimensional space to represent the original feature set of 12 values from a network packet.

The number of neurons are very important in impacting the outcome, and is unique to each dataset and context. It has to be considered thoroughly to avoid overfitting and underfitting of the model. To decide the number of neurons, a cross-validated grid-search was employed after a range for the number of neurons were found. Negated mean squared error (MSE) was used as the scoring metric to rank the lowest error the highest, capturing the difference between the input and output. The latent variable size was considered first. In Figure 4, the result of grid search done on latent space from 1 to 12 (size of input) is displayed. It is shown that after a latent space of 5, the improvement of achieving higher score diminishes, interpreted as additions of unnecessary computational load to the neural network. For the range of the size of the number of neurons, existing implementations in the field of autoencoders for network traffic with similar input features was reviewed, which scaled from 15 up to 200 [10, 11]. With these range in mind, a grid search has been employed with a set [20, 35, 50, 65, 100] for each hidden layer and a set [1, 2, 3, 4, 5] for the latent variable. The highest average score among 625 combinations of parameters possible, the structure [12, 65, 35, 20, 5, 20, 35, 65, 12] achieved the highest score of -0.118295 (result table found in Appendix A). The scores are very close among combinations with the same latent space. This is depicted in Figure 5 where the scores are clearly higher towards latent space 5 while the hidden layer combinations hugely depend on the latent space shown by each strip of data points plotted accordingly to the latent space.

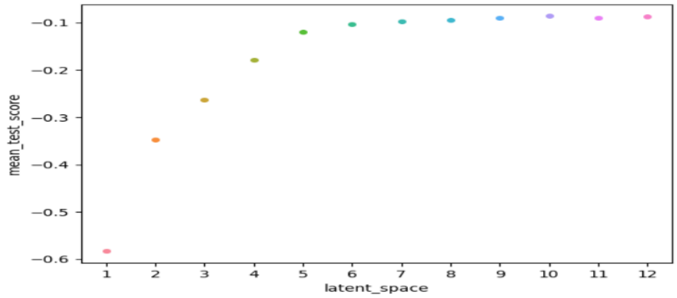


Figure 4: Latent space grid search

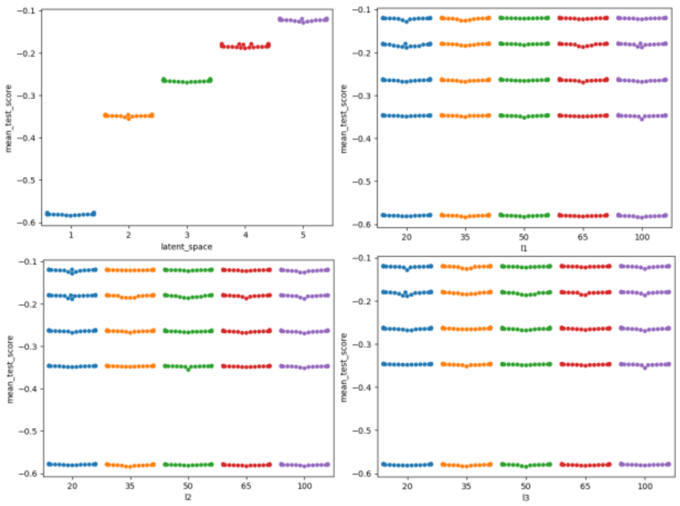


Figure 5: GridSearchCV plot

Activation function

Leaky ReLU was used as the activation function for the encoding hidden layers and linear (no activation function) in the decoding hidden layers. Leaky ReLU was used because of its faster and effective training of neural networks on large and complex datasets [12].

Loss function

For autoencoder models, the loss should describe how well the input has been reconstructed as output. Thus, to capture the difference of the two set, mean square error was used.

Optimizer

The Adam optimizer that combines best properties of several adaptive optimizer has been chosen. Adam optimizer can handle sparse and noisy data with the adaptive learning nature which suites for the network traffic dataset.

Batch size & epoch

The batch size has been set to 128. The epoch has been set to 25.

A full summary of this model can be found in Appendix B.

4.3 Preprocessing input and setup

For the input as the feature set that goes into the autoencoder, values such as IP address or prototype were label encoded into a value between zero and *n*-classes minus one as an approach of categorical encoding. This has been done since machine learning algorithms can only understand numbers, not texts, and the categorical difference in the values of IP address or prototype is the primary observation for machine learning algorithms, not the difference of exact textual values. To avoid weight bias coming from difference in relative range of values for input variables, each feature in the feature set were normalized into a value between zero and one using the MinMaxScaler. In Equation 1, x_{max} and x_{min} are the maximum and minimum value, respectively, in a column of feature set such as packet size and x_i is each one of the values in that feature set column.

$$\frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{1}$$

The autoencoder designed from subsection 4.2 run on the dataset from Table 1 on average of 30 runs gave a validation loss of 0.0181 with 70% and 30% used as the training and testing set, respectively. An example of an encoding process with the decoding is demonstrated in Table 2. The decoded representation shows the reconstruction for large values such as checksum or port numbers lacks performance, meanwhile the rest are reconstructed to the near tenths. The MSE with the normalized and decoded stage packet representation is 0.0142 with the following example.

Table 2: Example of packet encoding and decoding

Stage	Packet representation
Label encoded	[60, 1, 0, 16384, 6, 2, 23823, 0, 64, 35944, 50, 22422]
Normalized	[0, 1, 0, 1, 0, 0.33, 0.3635, 0, 0.1791, 0.7576, 0, 0.3422]
Encoded	[0.3772512, 0.14221649, -0.03652108, 0.31472498, 0.33470532]
Decoded	[0.0099, 0.9906, -0.0175, 0.9968, 0.0186, 0.3176, 0.0855, 0.0301, 0.1875, 0.7851, -0.0014, 0.6424]
Inverse transformed	[60.2763, 0.9906 -0.1578, 16331.375, 6.2045, 1.9054, 5605.1191, 5.7870, 64.5600, 37247.078, -40.203, 42097.836]

The experiments were conducted on a Lenovo X1 Carbon, equipped with an Intel Core i7 CPU, 2.60 GHz, with 16 GB RAM on Linux Ubuntu 20.04 LTS. Python TensorFlow and Keras were used to create and run the model for the autoencoder.

4.4 Evaluation metrics

To evaluate the results of the two versions of MalPaCA, quantitative evaluation metrics were used to compare the accuracy. The objective of MalPaCA is to cluster uni-directional connections into groups of similar behaviour on the network in order to give the cluster a label. Ideally, all the connections in a single cluster would have one label.

The quantitative metrics to indicate the accuracy of clusters will be malicious cluster purity, noise percentage and silhouette score. Malicious cluster purity is the percentage representing total number of 50% or higher of one malware label $(size_{mal_label})$ over the total cluster size excluding purely benign clusters shown in the equation below.

$$MCP = \frac{size_{mal\ label}}{\sum size_{all} - \sum size_{purely\ benign}} \tag{2}$$

This metric captures the accuracy for clustering malware by rewarding highly concentrated cluster of one malware label through penalizing clusters with mixed labels that do not have a single malware label that occurs more than 50% by adding a value of zero instead. Higher MCP grants higher confidence of labeling clusters. The noise percentage is calculated for the entire dataset expressed as Equation 3.

$$Noise\ percentage = \frac{\#\ of\ connections\ in\ noise\ cluster}{\#\ of\ total\ connections} \tag{3}$$

A lower noise percentage would indicate that more connections were assigned into clusters which are decided by the input to the clustering algorithm. As the input are values representing connections, if the values for connection that share very similar behaviour are smaller and connections that opposite behaviour are bigger, the noise percentage would be lower.

The labels listed in Table 1 of each connections were used in the experiment to deduce quantitative measures to help the evaluation for unsupervised learning of MalPaCA. For in depth analysis, qualitative measures such as temporal heatmaps are used. Temporal heatmaps displays the pattern existing in the sequential nature of the threshold value of packets.

5 Results

The baseline MalPaCA and AE integrated MalPaCA creates different sets of clusters, each showing unique characteristics of clusters of the network traffic. Few of the interesting behaviours captured by the two MalPaCA are described in this section.

5.1 Baseline MalPaCA

1. **Port scan identification**. Horizontal port scan connections are clustered into several clusters with 100% purity. Port scan is a technique used by hackers to find weak points in the network to proceed to break into a system. A horizontal port scan (HPS) attempts to make a connection through the same port with multiple different destination IP addresses. Destination port being one of the four feature, shown in Figure 6, all connections share the same destination port of 256 through all 20 packets. In addition, it has been detected that these HPS share the same packet size as well, shown in Figure 7.

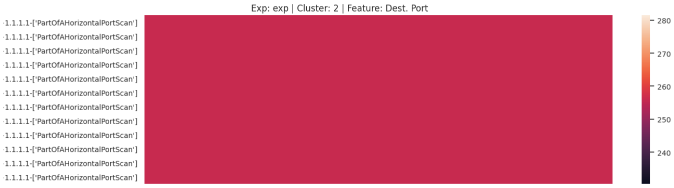


Figure 6: Baseline MalPaCA cluster 2 heat map - destination port

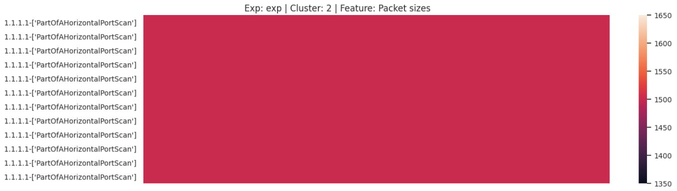


Figure 7: Baseline MalPaCA cluster 2 heat map - packet size

2. **C&C detection**. Infected devices connected to a C&C server are considered malicious connections labelled as C&C. With the baseline MalPaCA, C&C labelled connections are clustered purely. C&C labelled activities are assigned by evidence of connections to the suspicious IP being periodic found by the time interval from the feature set. The time interval is the time between two subsequent packets in a connection being sent. Figure 8 shows a pattern from a connection sending a group of packet very shortly after another followed by a break.

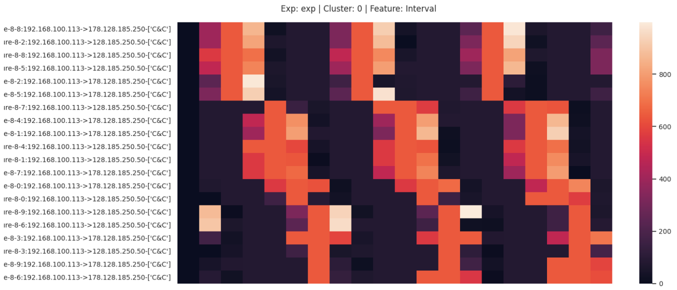


Figure 8: Baseline MalPaCA cluster 0 heat map - packet interval

3. **Malware behavioural clustering**. Cluster 14 is a group of DDoS and Okiru connections. MalPaCA captures the behaviours of these connection to group them together. As shown in Figure 9, both DDoS and Okiru connections have packet sizes constant throughout all 20 packets with minor exceptions. The packet interval of DDoS and Okiru are periodically equal with a pattern of being shorter every second packet shown in Figure 10. In addition, the destination port are identical with minor exceptions, seen in Figure 11. The similarity in behaviour gives a hint that the Okiru could be a type of DDoS attack.

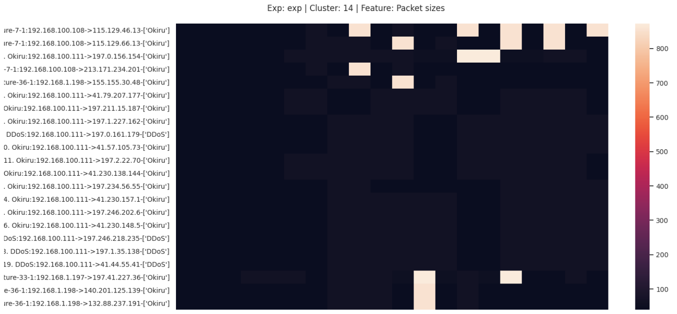


Figure 9: Baseline MalPaCA cluster 14 heat map - packet size

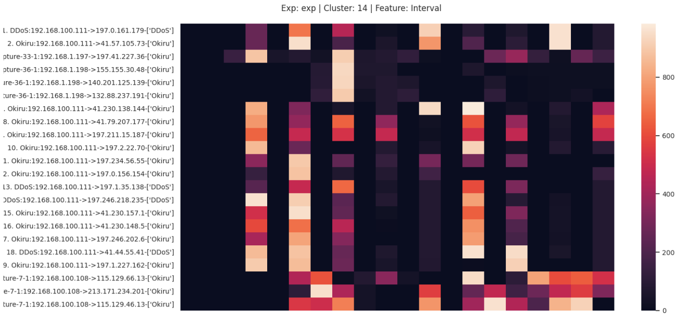


Figure 10: Baseline MalPaCA cluster 14 heat map - packet interval

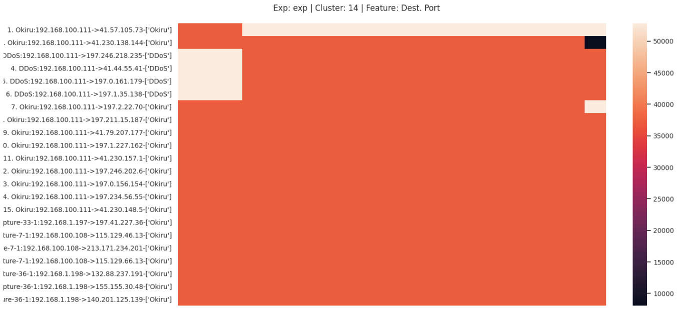


Figure 11: Baseline MalPaCA cluster 14 heat map - destination port

5.2 Autoencoder feature engineering MalPaCA

Before demonstrating the cluster characteristics using temporal heatmaps, it is worth mentioning that 5 heatmaps will be generated from 5 distance matrices due to the 5D latent space. The average MSE among the 5 distance matrices after L1 normalization was 3.97e-8, hence one of 5 heatmaps are used to the most visually distinguishable. The comparison of 5 heatmaps of a single cluster can be seen in Appendix C.

1. **Benign NTP connection detection**. Close to half of the benign connections were clustered in cluster 10. Figure 12 shows a snippet of the 473 connections temporal heatmap of which the full heatmap displays similar patterns. Packet encodings represented values increasing followed by a sudden decrease back to increasing depicted by the shading of colors on Figure 12. Inspecting these connections by Wireshark, all connections had packets consisting identical values except for

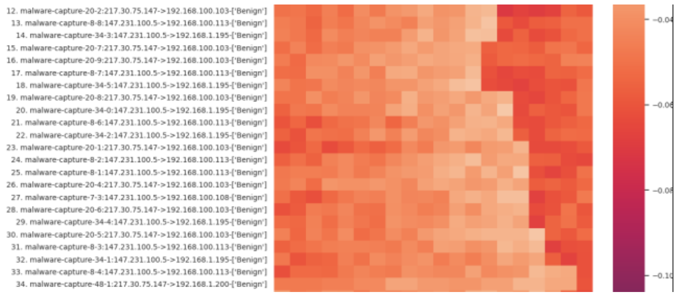


Figure 12: AE MalPaCA cluster 10 heat map (4D) (partial section)

the IP checksum, while IP addresses, TTL and TOS varied by connections. The increasing value of encoding were caused by the IP checksum field which gradually decreased until zero to jump back up to the highest. The protocol used were UDP with NTP. NTP is a built-on UDP using port 123 which is a port assigned by the Internet Assigned Numbers Authority [13]. NTP packets without any optional extensions, that may contain malicious load, are 76 bytes in size [14].

2. **DDoS attack detection**. DDoS attack were spotted into three different pure clusters accordingly to the type. Cluster 0, 2 and 5 contains the SYN flood, XMAS flood and UDP flood attack type of DDoS, respectively. These three clusters shared the characteristic of repeatedly sending the same packet from a single source IP shown in Figure 13. In the two clusters, the encoding of the packets in cluster 0 and 5 have intra-difference of 0.00003 and inter-difference of 0.23, implying the existence of two different set of identical packets. When inspected using Wireshark, the difference existed in packet size, IP destination, flag, and checksum while the rest of the fields remained identical.

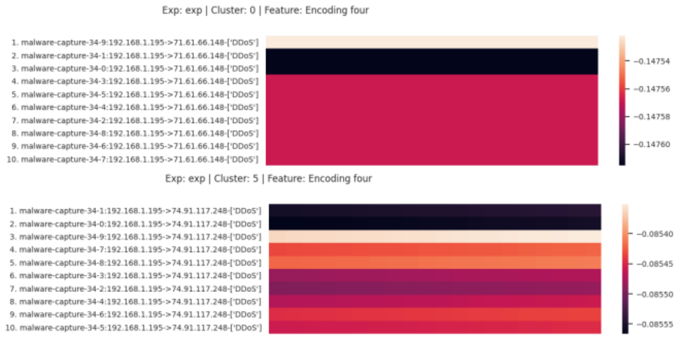


Figure 13: AE MalPaCA cluster 0 & 5 heat map (4D)

3. **C&C-Torii behaviour detection**. C&C-Torii connections were grouped together in two clusters by the direction of connection shown by green grids in Figure 14. C&C-Torii is an IoT botnet that stays stealthy and persistent to accomplish data exfiltration via multiple layers of encrypted communication. [15]. The heatmap shows C&C-Torii infected connections send identical packets over different intervals with one unique packet. Inspected in Wireshark, the unique packet were packets with different checksum and increased packet size to carry payload. The difference in the direction were captured by the difference in TTL, checksum and port number.

6 Comparison and discussion

The baseline MalPaCA and AE integrated MalPaCA, run on the same dataset and setting with the difference only at the extracted features, produced different characteristics of clusters. To compare the accuracy using the metrics from subsection 4.4 evaluating the performance, Table 3 has been created with the results from Appendix D.

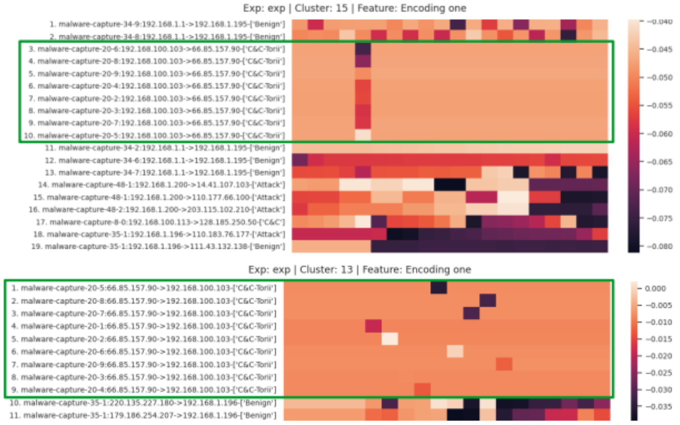


Figure 14: AE MalPaCA cluster 13 & 15 heat map (1D)

Table 3: Accuracy metrics from results of the two versions of MalPaCA

	MCP	Noise percentage	Silhouette score
Baseline MalPaCA	60.0%	9.3%	0.48
AE integrated MalPaCA	44.0%	20.3%	0.21

It is shown that the baseline MalPaCA outperforms the AE integrated MalPaCA in all metrics distinctively. A higher MCP and silhouette score allows the baseline MalPaCA to label malware clusters with higher confidence over the AE integrated MalPaCA. In addition, with less noise percentage, more connections from the dataset are conceivably able to be labeled. The higher MCP is a reflection of the AE integrated MalPaCA generating clusters with a high portion of benign connections mixed with malware such as in cluster 11 (poor feature extraction example shown by Figure 15) and clusters mixed of several malware connections that do not make the majority of the cluster such as in cluster 15 (poor clustering example shown by Figure 14). Malware such as 'C&C' and 'Part of a horizontal port scan' (HPS) were usually not clustered as a majority but as a minority in clusters instead of getting in the noise cluster, which is not the case with the baseline MalPaCA shown by cluster 0 (C&C), 12 (HPS) and 30 (HPS). Cluster 11 (Figure 15) can be seen as a result

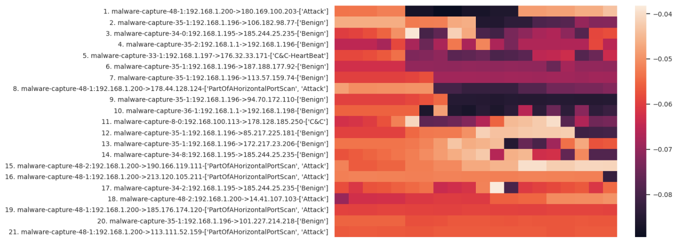


Figure 15: AE MalPaCA cluster 11 heatmap (1D)

of poor feature extraction, clustering 'Attack', 'C&C', HPS, 'C&C file download' and 'Benign' together. This can be explained by the initial input feature to the autoencoder. Despite the autoencoder's reconstruction error as low as 0.0181 (subsection 4.3), fields that help distinguish malware (i.e. C&C) such as interval (also used in baseline MalPaCA) or bit rate [11] were not included. Hence, correctly clustered connections were limited to connections that repeatedly send close to identical packets such as 'DDoS', 'Torii' and 'NTP benign' described in subsection 5.2.

In terms of explainability of clusters using the visual help of temporal heatmaps, baseline MalPaCA results contain more comprehensive and meaningful information. From baseline MalPaCA, the information gained from heatmaps directly assists the labelling of clusters in addition to accurate knowledge of the malware characteristic. Such examples are port scan scanning the same port to be characterized as a horizontal port scan (Figure 6 shows HPS at dst. port 256), and C&C sending a group of packets periodically (Figure 8). Contrarily, AE integrated MalPaCA heatmap interpretation is limited to pattern recognition, due to the values being an encoded. Although, it can be helpful to further inspect certain packets through Wireshark. Such as clusters with similar patterned connections, for example Figure 13 to find out the different shades represent different packet size, flag and checksum to spot different DDoS attack types. Or spotlight out-of-pattern packets for example Figure 14 hinting a packet with extra packet size.

7 Responsible Research

To reflect on the ethical aspects, privacy intrusion is considered the most for this research. Capturing the network traffic can become privacy intrusive. Intrusions to privacy is a paramount concern of research in the field of network measurement. Uninformed network monitoring become a problem regarding the privacy of the devices on that network.

For this research, privacy policy were adhered to the rules. The network traffic used in this research is generated by an antivirus software company, Avast, in a laboratory. The network traffic were captured in the laboratory under controlled network environment with unrestrained internet connection. Hence the issue of privacy intrusion of devices on the network is not violated. Furthermore, for the use of the MalPaCA algorithm with real network traffic, it is possible to follow the rules of privacy policies. Firstly, getting consent from the devices on the network will allow safe investigation of the network traffic. If getting consent of all devices is not trivial, using simulated data could replace [16]. Lastly, the advantage of MalPaCA is that the data used from the network traffic are limited to only the headers. Therefore, data reduction of packet information by cutting out the payload minimize legal exposure. Deep packet inspection is not carried out by MalPaCA.

The reproducibility of this research is positive due to the level of detail on the design choices. Throughout this paper, all descriptions and designs of the experiment is clearly written, and in particular, section 4 provides the guideline to set up the experiment. The dataset used is under the references to be found.

8 Conclusions and Future Work

An automated feature engineering for the MalPaCA algorithm has been attempted to improve the performance of malware capability assessment to build network behavioral profiles replacing the baseline version's hand-crafted feature set. The baseline MalPaCA successfully builds network behavioral profiles by separating different families of malware into different clusters, while the AE integrated MalPaCA is limited to building network profiles of certain type of malware (i.e. malware sending repeated close to identical packets), resulting in clusters with multiple labels. This is caused by the the AE generated feature set that has encoded features from 12 fields of the packet header that lacks the ability to detect certain malware types. In conclusion, both accuracy and explainability of the clusters have been outperformed by the baseline MalPaCA.

Performance of the autoencoder integrated MalPaCA has areas for improvement. One area is in improving the feature extraction by inclusion of features known to detect malware. As mentioned in section 6, malwares such as C&C and HPS were finely clustered in the baseline MalPaCA with clear patterns in the feature time interval. The other area is in improving the clustering algorithm. There were groups of connections within single clusters with clear similarities such as in Figure 14 and Figure 15. These groups of connections would be ideal to have its own cluster.

References

[1] Asma Zahra and Munam Ali Shah. "IoT based ransomware growth rate evaluation and detection using command and control blacklisting". In: *2017 23rd International Conference on Automation and Computing (ICAC)*. Sept. 2017, pp. 1–6. DOI: 10.23919/IConAC.2017.8082013.

[2] Arushi Sharma et al. "Malware Capability Assessment using Fuzzy Logic". In: *Cybernetics and Systems* 50.4 (May 2019), pp. 323–338. DOI: 10.1080/01969722.2018.1552906. URL: https://doi.org/10.1080/01969722.2018.1552906.

[3] Azqa Nadeem et al. "Beyond Labeling: Using Clustering to Build Network Behavioral Profiles of Malware Families". In: *Malware Analysis Using Artificial Intelligence and Deep Learning*. Ed. by Mark Stamp, Mamoun Alazab, and Andrii Shalaginov. Springer International Publishing, 2021, pp. 381–409. ISBN: 9783030625825. DOI: 10.1007/978-3-030-62582-5_15. URL: https://doi.org/10.1007/978-3-030-62582-5_15.

[4] Ahmad Azab, Mamoun Alazab, and Mahdi Aiash. "Machine learning based botnet identification traffic". In: *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE. 2016, pp. 1788–1794.

[5] Leyla Bilge et al. "Disclosure: detecting botnet command and control servers through large-scale netflow analysis". In: *Proceedings of the 28th Annual Computer Security Applications Conference*. 2012, pp. 129–138.

[6] Mayu Sakurada and Takehisa Yairi. "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction". In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis - MLSDA'14*. ACM Press, 2014. DOI: 10.1145/2689746.2689747. URL: https://doi.org/10.1145/2689746.2689747.

[7] Sebastian Garcia, Agustin Parmisano, and Maria Jose Erquiaga. *IoT-23: A labeled dataset with malicious and benign IoT network traffic*. Jan. 2020. DOI: 10.5281/zenodo.4743746. URL: https://zenodo.org/record/4743746.

[8] Lidong Wang and Randy Jones. "Big Data Analytics of Network Traffic and Attacks". In: *NAECON 2018-IEEE National Aerospace and Electronics Conference*. IEEE. 2018, pp. 117–123.

[9] D. Stathakis. "How many hidden layers and nodes?" In: *International Journal of Remote Sensing* 30.8 (Apr. 2009), pp. 2133–2147. DOI: 10.1080/01431160802549278. URL: https://www.tandfonline.com/doi/full/10.1080/01431160802549278.

[10] Mahmood Yousefi-Azar et al. "Autoencoder-based feature learning for cyber security applications". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. May 2017, pp. 3854–3861. DOI: 10.1109/IJCNN.2017.7966342.

[11] Gianni D'Angelo and Francesco Palmieri. "Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction". In: *Journal of Network and Computer Applications* 173 (Jan. 2021), p. 102890. DOI: 10.1016/j.jnca.2020.102890. URL: https://doi.org/10.1016/j.jnca.2020.102890.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[13] John R. Vacca. *Managing information security*. Elsevier, 2014. URL: https://www-sciencedirect-com.tudelft.idm.oclc.org/science/article/pii/B9780124166882000064.

[14] Geoff Huston. *NTP for Evil*. Mar. 2014. URL: https://labs.apnic.net/?p=464.

[15] Jakub Kroustek et al. *New Torii Botnet uncovered, more sophisticated than Mirai*. URL: https://blog.avast.com/new-torii-botnet-threat-research.

[16] Paul Ohm, Douglas Sicker, and Dirk Grunwald. "Legal Issues Surrounding Monitoring During Network Research (Invited Paper)". In: (Jan. 2007).

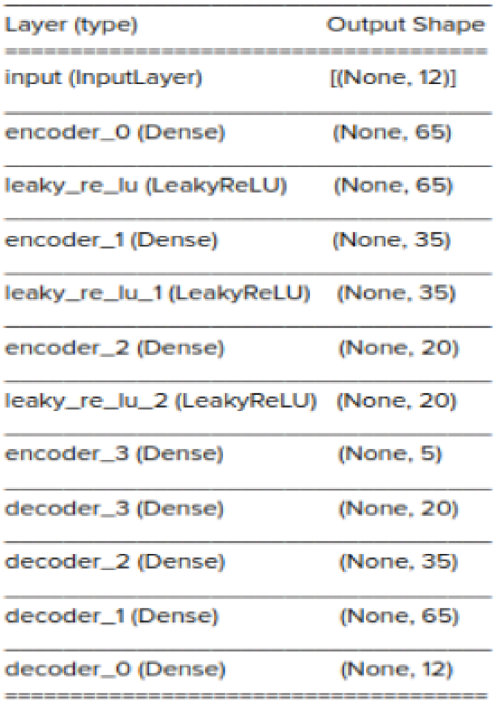
A Cross validated grid search result

Table 4: Grid search score of top 100 hyper parameter combination

Mean test score	Std. test score	Parameters
-0.118295	0.166316	{’l1’: 65, ’l2’: 35, ’l3’: 20, ’latent_space’: 5}
-0.118438	0.166236	{’l1’: 100, ’l2’: 50, ’l3’: 35, ’latent_space’: 5}
-0.118442	0.165742	{’l1’: 65, ’l2’: 50, ’l3’: 35, ’latent_space’: 5}
-0.118496	0.166094	{’l1’: 100, ’l2’: 50, ’l3’: 50, ’latent_space’: 5}
-0.118503	0.166477	{’l1’: 100, ’l2’: 35, ’l3’: 65, ’latent_space’: 5}
-0.118592	0.165745	{’l1’: 50, ’l2’: 20, ’l3’: 100, ’latent_space’: 5}
-0.118611	0.166284	{’l1’: 65, ’l2’: 50, ’l3’: 65, ’latent_space’: 5}
-0.118638	0.165994	{’l1’: 65, ’l2’: 50, ’l3’: 20, ’latent_space’: 5}
-0.118651	0.166248	{’l1’: 65, ’l2’: 20, ’l3’: 65, ’latent_space’: 5}
-0.118658	0.166377	{’l1’: 50, ’l2’: 65, ’l3’: 50, ’latent_space’: 5}
-0.118668	0.165862	{’l1’: 65, ’l2’: 35, ’l3’: 65, ’latent_space’: 5}
-0.118671	0.165804	{’l1’: 100, ’l2’: 20, ’l3’: 50, ’latent_space’: 5}
-0.118733	0.166101	{’l1’: 100, ’l2’: 20, ’l3’: 100, ’latent_space’: 5}
-0.118749	0.166334	{’l1’: 35, ’l2’: 65, ’l3’: 35, ’latent_space’: 5}
-0.118772	0.166404	{’l1’: 35, ’l2’: 35, ’l3’: 35, ’latent_space’: 5}
-0.118804	0.166124	{’l1’: 20, ’l2’: 100, ’l3’: 20, ’latent_space’: 5}
-0.118809	0.165830	{’l1’: 35, ’l2’: 50, ’l3’: 20, ’latent_space’: 5}
-0.118814	0.166135	{’l1’: 50, ’l2’: 100, ’l3’: 35, ’latent_space’: 5}
-0.118818	0.165986	{’l1’: 65, ’l2’: 35, ’l3’: 100, ’latent_space’: 5}
-0.118820	0.166109	{’l1’: 65, ’l2’: 100, ’l3’: 65, ’latent_space’: 5}
-0.118839	0.166008	{’l1’: 50, ’l2’: 65, ’l3’: 20, ’latent_space’: 5}
-0.118842	0.166542	{’l1’: 50, ’l2’: 50, ’l3’: 20, ’latent_space’: 5}
-0.118881	0.166286	{’l1’: 65, ’l2’: 20, ’l3’: 35, ’latent_space’: 5}
-0.118895	0.165867	{’l1’: 35, ’l2’: 20, ’l3’: 50, ’latent_space’: 5}
-0.118904	0.166292	{’l1’: 100, ’l2’: 100, ’l3’: 20, ’latent_space’: 5}
-0.118915	0.166708	{’l1’: 50, ’l2’: 50, ’l3’: 100, ’latent_space’: 5}
-0.118938	0.166645	{’l1’: 50, ’l2’: 20, ’l3’: 65, ’latent_space’: 5}
-0.118946	0.166378	{’l1’: 50, ’l2’: 100, ’l3’: 100, ’latent_space’: 5}
-0.118953	0.166768	{’l1’: 35, ’l2’: 20, ’l3’: 65, ’latent_space’: 5}
-0.118965	0.166383	{’l1’: 100, ’l2’: 50, ’l3’: 100, ’latent_space’: 5}
-0.118970	0.166404	{’l1’: 50, ’l2’: 20, ’l3’: 35, ’latent_space’: 5}
-0.118972	0.166153	{’l1’: 50, ’l2’: 50, ’l3’: 35, ’latent_space’: 5}
-0.118974	0.166082	{’l1’: 35, ’l2’: 65, ’l3’: 100, ’latent_space’: 5}
-0.118978	0.166209	{’l1’: 35, ’l2’: 20, ’l3’: 20, ’latent_space’: 5}
-0.118984	0.165783	{’l1’: 100, ’l2’: 20, ’l3’: 65, ’latent_space’: 5}
-0.118995	0.165897	{’l1’: 50, ’l2’: 100, ’l3’: 65, ’latent_space’: 5}
-0.118995	0.166504	{’l1’: 20, ’l2’: 65, ’l3’: 35, ’latent_space’: 5}
-0.119001	0.166093	{’l1’: 20, ’l2’: 20, ’l3’: 100, ’latent_space’: 5}
-0.119018	0.166081	{’l1’: 20, ’l2’: 35, ’l3’: 50, ’latent_space’: 5}
-0.119029	0.166986	{’l1’: 50, ’l2’: 35, ’l3’: 100, ’latent_space’: 5}
-0.119030	0.166490	{’l1’: 100, ’l2’: 100, ’l3’: 100, ’latent_space’: 5}
-0.119031	0.166126	{’l1’: 50, ’l2’: 50, ’l3’: 65, ’latent_space’: 5}
-0.119048	0.166292	{’l1’: 65, ’l2’: 65, ’l3’: 50, ’latent_space’: 5}
-0.119110	0.166192	{’l1’: 35, ’l2’: 100, ’l3’: 50, ’latent_space’: 5}
-0.119151	0.165886	{’l1’: 65, ’l2’: 20, ’l3’: 100, ’latent_space’: 5}
-0.119212	0.165871	{’l1’: 50, ’l2’: 65, ’l3’: 35, ’latent_space’: 5}
-0.119225	0.165761	{’l1’: 100, ’l2’: 35, ’l3’: 100, ’latent_space’: 5}
-0.119237	0.166184	{’l1’: 100, ’l2’: 20, ’l3’: 35, ’latent_space’: 5}
-0.119262	0.166297	{’l1’: 35, ’l2’: 50, ’l3’: 65, ’latent_space’: 5}
-0.119336	0.165473	{’l1’: 100, ’l2’: 65, ’l3’: 100, ’latent_space’: 5}
-0.119337	0.165768	{’l1’: 35, ’l2’: 100, ’l3’: 20, ’latent_space’: 5}
-0.119382	0.166676	{’l1’: 35, ’l2’: 35, ’l3’: 50, ’latent_space’: 5}
-0.119398	0.167847	{’l1’: 50, ’l2’: 65, ’l3’: 65, ’latent_space’: 5}
-0.119438	0.166904	{’l1’: 65, ’l2’: 65, ’l3’: 35, ’latent_space’: 5}
-0.119441	0.165923	{’l1’: 100, ’l2’: 20, ’l3’: 20, ’latent_space’: 5}
-0.119457	0.165794	{’l1’: 20, ’l2’: 35, ’l3’: 20, ’latent_space’: 5}
-0.119519	0.166590	{’l1’: 20, ’l2’: 35, ’l3’: 35, ’latent_space’: 5}
-0.119530	0.166065	{’l1’: 100, ’l2’: 65, ’l3’: 50, ’latent_space’: 5}
-0.119530	0.165730	{’l1’: 50, ’l2’: 50, ’l3’: 50, ’latent_space’: 5}

-0.119538	0.166362	{’l1’: 20, ’l2’: 35, ’l3’: 100, ’latent_space’: 5}
-0.119552	0.165689	{’l1’: 50, ’l2’: 100, ’l3’: 50, ’latent_space’: 5}
-0.119555	0.166722	{’l1’: 65, ’l2’: 20, ’l3’: 50, ’latent_space’: 5}
-0.119592	0.165918	{’l1’: 50, ’l2’: 100, ’l3’: 20, ’latent_space’: 5}
-0.119665	0.166151	{’l1’: 100, ’l2’: 65, ’l3’: 20, ’latent_space’: 5}
-0.119698	0.165737	{’l1’: 65, ’l2’: 65, ’l3’: 65, ’latent_space’: 5}
-0.119759	0.166951	{’l1’: 65, ’l2’: 65, ’l3’: 100, ’latent_space’: 5}
-0.119766	0.165471	{’l1’: 65, ’l2’: 35, ’l3’: 35, ’latent_space’: 5}
-0.119779	0.165797	{’l1’: 20, ’l2’: 65, ’l3’: 65, ’latent_space’: 5}
-0.119807	0.167608	{’l1’: 20, ’l2’: 100, ’l3’: 65, ’latent_space’: 5}
-0.119828	0.165536	{’l1’: 20, ’l2’: 50, ’l3’: 20, ’latent_space’: 5}
-0.119830	0.167227	{’l1’: 50, ’l2’: 35, ’l3’: 65, ’latent_space’: 5}
-0.119853	0.165483	{’l1’: 100, ’l2’: 35, ’l3’: 50, ’latent_space’: 5}
-0.119917	0.165650	{’l1’: 65, ’l2’: 35, ’l3’: 50, ’latent_space’: 5}
-0.119917	0.168588	{’l1’: 20, ’l2’: 50, ’l3’: 100, ’latent_space’: 5}
-0.119939	0.165614	{’l1’: 65, ’l2’: 50, ’l3’: 100, ’latent_space’: 5}
-0.120026	0.165565	{’l1’: 50, ’l2’: 35, ’l3’: 35, ’latent_space’: 5}
-0.120036	0.165497	{’l1’: 35, ’l2’: 50, ’l3’: 35, ’latent_space’: 5}
-0.120049	0.165622	{’l1’: 20, ’l2’: 65, ’l3’: 20, ’latent_space’: 5}
-0.120061	0.165510	{’l1’: 65, ’l2’: 65, ’l3’: 20, ’latent_space’: 5}
-0.120090	0.165560	{’l1’: 50, ’l2’: 65, ’l3’: 100, ’latent_space’: 5}
-0.120162	0.165920	{’l1’: 100, ’l2’: 100, ’l3’: 35, ’latent_space’: 5}
-0.120171	0.165910	{’l1’: 20, ’l2’: 50, ’l3’: 65, ’latent_space’: 5}
-0.120186	0.166324	{’l1’: 20, ’l2’: 50, ’l3’: 35, ’latent_space’: 5}
-0.120222	0.167728	{’l1’: 100, ’l2’: 50, ’l3’: 65, ’latent_space’: 5}
-0.120278	0.166278	{’l1’: 35, ’l2’: 35, ’l3’: 65, ’latent_space’: 5}
-0.120279	0.166675	{’l1’: 20, ’l2’: 100, ’l3’: 100, ’latent_space’: 5}
-0.120336	0.167012	{’l1’: 65, ’l2’: 50, ’l3’: 50, ’latent_space’: 5}
-0.120441	0.165803	{’l1’: 35, ’l2’: 100, ’l3’: 65, ’latent_space’: 5}
-0.120474	0.165605	{’l1’: 50, ’l2’: 35, ’l3’: 50, ’latent_space’: 5}
-0.120506	0.165934	{’l1’: 50, ’l2’: 20, ’l3’: 20, ’latent_space’: 5}
-0.120613	0.165730	{’l1’: 100, ’l2’: 100, ’l3’: 65, ’latent_space’: 5}
-0.120636	0.166345	{’l1’: 35, ’l2’: 65, ’l3’: 50, ’latent_space’: 5}
-0.120637	0.165554	{’l1’: 20, ’l2’: 20, ’l3’: 50, ’latent_space’: 5}
-0.120672	0.165346	{’l1’: 20, ’l2’: 65, ’l3’: 50, ’latent_space’: 5}
-0.120678	0.165310	{’l1’: 100, ’l2’: 35, ’l3’: 35, ’latent_space’: 5}
-0.120729	0.165080	{’l1’: 100, ’l2’: 35, ’l3’: 20, ’latent_space’: 5}
-0.120780	0.169519	{’l1’: 35, ’l2’: 65, ’l3’: 65, ’latent_space’: 5}
-0.120817	0.165660	{’l1’: 35, ’l2’: 35, ’l3’: 100, ’latent_space’: 5}
-0.120818	0.165404	{’l1’: 20, ’l2’: 100, ’l3’: 35, ’latent_space’: 5}
-0.120826	0.166346	{’l1’: 20, ’l2’: 35, ’l3’: 65, ’latent_space’: 5}

B Model summary



```
_________________________________________________
Layer (type)                      Output Shape
=================================================
input (InputLayer)                [(None, 12)]
_________________________________________________
encoder_0 (Dense)                 (None, 65)
_________________________________________________
leaky_re_lu (LeakyReLU)           (None, 65)
_________________________________________________
encoder_1 (Dense)                 (None, 35)
_________________________________________________
leaky_re_lu_1 (LeakyReLU)         (None, 35)
_________________________________________________
encoder_2 (Dense)                 (None, 20)
_________________________________________________
leaky_re_lu_2 (LeakyReLU)         (None, 20)
_________________________________________________
encoder_3 (Dense)                 (None, 5)
_________________________________________________
decoder_3 (Dense)                 (None, 20)
_________________________________________________
decoder_2 (Dense)                 (None, 35)
_________________________________________________
decoder_1 (Dense)                 (None, 65)
_________________________________________________
decoder_0 (Dense)                 (None, 12)
=================================================
```

Figure 16: Autoencoder model summary from TensorFlow

C 5 heatmaps of same cluster

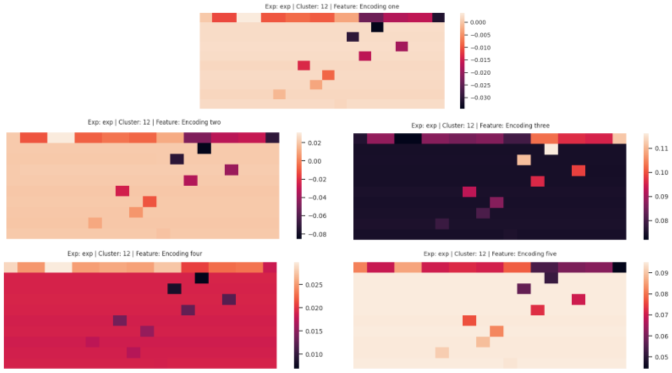


Figure 17: Heatmap of all 5 encoding spaces of cluster 12 showing same patterns

D MalPaCA cluster results

D.1 Baseline MalPaCA cluster results

Table 5: Baseline MalPaCA cluster results

Cluster	Num. of conn	Labels with distribution
0	20	’C&C’:100%
1	18	’C&C’:11.1%, ’C&C-heartbeat-file-download’:11.1%, ’C&C-file-download’:5.5%, ’Benign’:72.3%
2	12	’Part-of-a-horizontal-port-scan’:100%
3	12	’DDoS’:100%
4	46	’Attack’:30.4%, ’Part-of-a-horizontal-port-scan’:10.9%, ’Benign’:50.7%
5	10	’C&C-Torii’:100%
6	10	’C&C-Torii’:100%
7	10	’C&C-Torii’:100%
8	10	’C&C-Torii’:100%
9	83	’Attack’:27.7%, ’Part-of-a-horizontal-port-scan’:14.5%, ’Benign’:57.8%
10	13	’Part-of-a-horizontal-port-scan’:7,7%, ’Benign’:92.3%
11	20	’Benign:100%
12	23	’DDoS’:8.7%, ’File-download’:4.3%, ’Benign’:87%
13	11	’Okiru’9.1%, ’Benign’:90.9%
14	22	’DDoS’:18.2%, ’Okiru’:81.2%
15	18	’File-download’:5.6%, ’Benign’:94.4%
16	19	’Benign’:100%
17	26	’Benign’:100%
18	41	’Benign’:100%
19	55	’Benign’:100%
20	75	’Benign’:100%
21	14	’Benign’:100%
22	84	’Benign’:100%
23	50	’Benign’:100%
24	19	’Benign’:100%
25	57	’Benign’:100%
26	74	’Benign’:100%
27	37	’Benign’:100%
28	12	’Benign’:100%
29	12	’Benign’:100%
30	12	’Part-of-a-horizontal-port-scan’:100%
31	12	’DDoS’:100%
32	12	’DDoS’:100%
-1	111	’DDoS’:17.1%, ’C&C’:1.8%, ’C&C-heartbeat-File-download’:1.8%, ’C&C-heartbeat’:9.9%, ’C&C-File-download’:0.1%, ’Part-of-a-horizontal-port-scan’:2.7%, ’Okiru’:1.8%, ’Benign’:64.8%

D.2 Autoencoder integrated MalPaCA cluster results

Cluster	Num. of conn	Labels with distribution
0	10	’DDoS’:100%
1	10	’Benign’:100%
2	10	”DDoS’:100%
3	45	’DDoS’:17.8%, ’Part-of-a-horizontal-port-scan’:22.2%, ’Okiru’:33.3%, ’Benign’:26.7%
4	52	’DDoS’:7.7%, ’Benign’:92.3%
5	10	’DDoS’:100%
6	10	’Part-of-a-horizontal-port-scan’:100%
7	33	’Part-of-a-horizontal-port-scan’:1.5% , ’Benign’:98.5%
8	17	’DDoS’:11.8%, ’Benign’:88.2%
9	32	’Benign’:100%
10	473	’Benign’:100%
11	16	’Attack’:6.3%, ’Part-of-a-horizontal-port-scan’:6.3%, ’C&C’:3.2%, ’C&C-file-download’:3.2%, ’C&C-Torii’:6.3%, ’Benign’:74.7%
12	10	’C&C-Torii’:90% , ’Benign’:10%
13	11	’C&C-Torii’:81.8%, ’Benign’:18.2%
14	21	’Part-of-a-horizontal-port-scan’:23.8%, ’Attack’:9.5%, ’C&C’:5.3%, ’C&C-heartBeat’:5.3%, ’Benign’:56.1%
15	19	’Attack’:21.1%, ’C&C-Torii’:42.1%, ’C&C’:5.3%, ’Benign’:31.5%

-1	235	’DDoS’:4.7%, ’C&C-Torii’:5.5%, ’C&C’:9.4%, ’Attack’:12.8 ’C&C-heartbeat-file-download’:1.7%, ’Part-of-a-horizontal-port-scan’:6.0%, ’C&C-heartbeat’:4.3%, ’File-download’:0.1%, ’C&C-file-download’:0.1%, ’Okiru’:2.6%, ’Benign’:52.8%