Pay Attention: Improving Classification of PE Malware Using Attention Mechanisms Based on System Call Analysis

Ori Or-Meir,[a,b] Aviad Cohen,[a] Yuval Elovici,[a,b] Lior Rokach,[a,b] Nir Nissim,[a,c]

[a] *Malware Lab, Cyber Security Research Center, Ben-Gurion University of the Negev, Israel*

[b] *Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Israel*

[c] *Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Israel*

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533481

Abstract.

Malware poses a threat to computing systems worldwide, and security experts work tirelessly to detect and classify malware as accurately and quickly as possible. Since malware can use evasion techniques to bypass static analysis and security mechanisms, dynamic analysis methods are more useful for accurately analyzing the behavioral patterns of malware. Previous studies showed that malware behavior can be represented by sequences of executed system calls and that machine learning algorithms can leverage such sequences for the task of malware classification (a.k.a. malware categorization). Accurate malware classification is helpful for malware signature generation and is thus beneficial to antivirus vendors; this capability is also valuable to organizational security experts, enabling them to mitigate malware attacks and respond to security incidents. In this paper, we propose an improved methodology for malware classification, based on analyzing sequences of system calls invoked by malware in a dynamic analysis environment. We show that adding an attention mechanism to a LSTM model improves accuracy for the task of malware classification, thus outperforming the state-of-the-art algorithm by up to 6%. We also show that the transformer architecture can be used to analyze very long sequences with significantly lower time complexity for training and prediction. Our proposed method can serve as the basis for a decision support system for security experts, for the task of malware categorization.

1. Introduction

Malware has been around since the early days of computing. Antivirus (AV) software is used by most organizations, governments, and companies worldwide to detect the many risks malware poses, in order to mitigate it. Such risks include sensitive data exfiltration; denial of services and critical data (e.g., CryptoLockers; ransomware) [1], diverting a user's computational resources for an attacker's benefit (cryptojacking) [2], and more. Most AV software relies on deterministic and static signatures extracted from known malware to detect the known malware samples and their new variants. Although most AVs provide an adequate level of protection against known malware, they are limited in their ability to adapt to the fast-paced world in which new malware is continuously created with new modus operandi and evasion techniques that limit their detection. When a new malware that has spread and caused damage is discovered by an AV vendor or cyber-security company, a signature must be extracted immediately from the malware's binary file and shared with all the clients and users of the AV vendor. This process can be time-consuming, creating an open door for malicious parties that want to exploit vulnerabilities and attack vulnerable computational systems. In addition, detection using signatures is limited by various static evasion mechanisms employed by malware to avoid detection, such as code obfuscation, polymorphism, and dynamic code loading [3]. To reduce reliance on up-to-date static signatures, malware can be also detected by its runtime behavior. The process of executing and analyzing malware behavior in a controlled environment to identify behavioral patterns is called **dynamic malware analysis** [4]. By analyzing malware behavior in real time, it is possible to overcome the static evasion techniques mentioned earlier and obtain increased understanding regarding the behavior of the examined application; moreover, it is also possible to generate a behavioral pattern that can either be used for malware detection or classification. Having a fast and reliable method for classifying malware is essential for security experts and AV vendors. Knowing which type of malware we are dealing with can be helpful in multiple ways; it can expedite generating a malware signature and updating the antivirus to protect vulnerable systems, and when an organization has been the victim of a malware attack, knowing which type of malware it is can help security experts respond more quickly and accurately to mitigate the threat.

In this paper, we propose a new, automated malware classification method that is based on analyzing sequences of system calls invoked during the execution of the examined malware. Such sequences represent the core functionality of a malware behavior at the system call level and can be used to classifying malware by type. Our method leverages both attention [5] and transformer architecture [6] to classify a sequence of system calls executed by malware to the correct malware type. We show that our proposed method outperformed a model based on the long short-term memory (LSTM) architecture [7] for the task of malware classification.

The contributions of this paper are as follows:

- We are the first to propose the application of attention and transformer architecture to the analysis of system call sequences for unknown malware classification.
- We are the first to explore and shed light on the influence of the system call sequence's length on both malware classification accuracy and performance time (training and classifying).
- We show that the use of transformer architecture can save up to 75% of the training and prediction time, which makes our proposed solution an applicable malware classification tool.

2. Background

a) Malware and System Calls

Modern operating systems have a separation between user-mode code and kernel-mode code. This separation is key to the enforcement of security protocols, user space separation, preventing race conditions between different processes or services, and more. Kernel-mode code has more privileges than user-mode code, including direct access to physical hardware, control of RAM allocations, read/write access to the hard drive, etc.

The basic functionality of any operating system is executed in kernel-mode as well as hardware device drivers (e.g., network interface cards, hard drives, audio). The user mode is allocated to the higher-level functions of the operating system and most applications installed by the user [8]. When a user-mode process requires some kernel-mode functions, a switch in operating mode must occur within the CPU. This switch cannot be performed without permissions, and the control must be handed over to the kernel in an orderly manner. To perform this handover of control between user-mode to kernel-mode, special kind of functions must be invoked. Such function invocations are known as **system calls**. For example, in the Windows 7 operating system, the list of system calls includes *NtAllocateVirtualMemory* (for memory allocation), *NtFreeVirtualMemory* (for releasing memory when it is not needed anymore), *NtOpenProcess* (to create a new process), *NtOpenFile* (to open a file for reading and/or writing), *NtDeleteFile* (to delete a file), and many others.

When dynamically analyzing malware, we expect to see many events happenings within a short period of time: files are read and written to, data is sent over the network, queries are executed to the operating system, and messages are sent to other processes. While all malware samples behave differently, each must interact with the operating system and the physical hardware by invoking system calls. Since each operating system has a few hundred system calls, which is a relatively low number compared to the complexity of modern processes, our hypothesis is that the order of the system calls invoked during runtime can be used to identify the behavior of the malware and therefore can help in malware attribution.

b) Deep Learning

Artificial neural networks (ANN) form a family of machine learning algorithms that introduced a new mechanism for learning [9] in which data is fed into an input layer of artificial computation units (called 'neurons') and propagates through layers of such neurons. The output (numerical, categorical, or any other type of output) is produced in the last layer. A neural network model composed of multiple layers of various types of neuron layers is called a **deep neural network** (DNN). The more layers the network has, the 'deeper' it is. The number of layers and the type of layers employed are selected to handle a specific task, with the general assumption that 'deeper' neural networks can learn more complex connections and correlations in the input data and can even learn features automatically (a.k.a. representation learning [10]).

c) Sequence Analysis with Artificial Neural Networks

When the input data is an ordered sequence of items or events, DNNs are limited in their ability to learn the context of an item. For this task **recurrent neural networks** (RNNs) were developed [11]–[13]. An RNN is composed of artificial neurons which are 'deep in time' rather than 'deep in space,' meaning that for each item S_i in the sequence S, the output of the RNN $\widehat{S_i}$ is fed to the RNN, along with the next item in the sequence (S_{i+1}). This allows the artificial network to learn the 'context' of each input based on the previous (or latter) data in that sequence. Since the RNN unit sees each item in the sequence and applies its output to the next item, we say that this type of neural network is 'deep in time.' RNNs have been used successfully in the fields of natural language processing (NLP) and can be used to autocomplete, translate, and manipulate a given text [14], [15].

While RNNs work as expected for short sequences, as the input sequence grows longer, the RNN units starts to 'forget' the input from the beginning of the sequence. This phenomenon is known as the *diminishing gradient* [16]. Due to the complexity of very long sequences, RNNs struggle to analyze and correlate items very far apart in the sequence. Two RNN variants were developed to address this issue: **gated recurrent units** (GRUs) and **long short-term memory** (LSTM). Research has shown that GRUs and LSTM have comparable performance although in some instances a GRU can be faster to train [17].

Attention mechanisms [5] allow modeling dependencies between sequence items, regardless of their distance from one another in the input, thus minimizing the vanishing gradient problem of RNNs. This is accomplished by connecting the output of each RNN step to a fully connected layer. The weights of these connections represent the correlation between items in the same sequence. Since each item can have a strong or weak connection to all other items, the attention layer allows the model to learn which subsequences of a given input better correlate to a specific prediction.

Bahdanau et al. [5] showed that **bidirectional RNNs** can be applied for the task of natural machine translation, with the added value of an attention mechanism for translating longer sentences. Their paper demonstrated that for translation purposes, it is more useful to have an attention mechanism, as it allows the model to focus on specific parts of a given sentence, much like a person does when translating. Other prior studies have has applied attention mechanisms for various problems, such as text translation, sentiment analysis, and image captioning [18]–[20].

A transformer is a neural network architecture for sequence analysis that relies solely on attention mechanisms, without using any type of RNN [6]. Basically, without having a recurrent unit, the transformer cannot distinguish the order of items in the sequence. To cope with this, the position of each item is fed into an embedding layer to create a positional encoding. The transformer architecture was first developed for natural language processing (NLP), but it was later applied to various problems, such as radio signal analysis [21], face alignment in videos [22], and more.

3. Related work

This section presents a variety of studies that introduced novel algorithms and methods using system call sequences for the tasks of malware detection and malware classification. Our goal is to understand the techniques used and identify the gaps and use this knowledge to improve existing state-of-the-art methods.

One way of dealing with system call sequences is by counting the number of times each system call was executed by each sample. The algorithm developed to accomplish this is known as the bag of words algorithm, and it can be very efficient for training and classifying [23]. However, since the order and context of the system calls are not considered, the algorithm cannot identify complex malicious behavior over time, as is required for accurate malware classification. Overall, sequential data contains more information than data removed from its sequential context and so provides additional information for difficult classification tasks, such as identifying the malware type.

Several types of sequence-based machine learning algorithms have been used for malware detection or classification. The earliest research, as far as we could determine, was published in 1999 [24], presenting a malware detection algorithm using the hidden Markov model (HHM). Since then, other machine learning algorithms, including SVM [23], [25] and stochastic gradient descend [26], have been presented. Such approaches are limited in their ability to learn from long sequences, as they make predictions based on recent events, and they are incapable of capturing the wide context of system call sequences.

With the recent rise of neural networks, studies have shown that several architectures can be applied to enhance malware detection using system call sequences: RNNs [27], [28], **convolutional neural networks** (CNNs) [27], [29], and **autoencoders** (AEs) [30], [31]. Pascanu et al. [28] trained an RNN to predict the next system call for a given sequence. Using the hidden states of their trained model as a feature vector, they trained a separate classifier to detect malware.

Tobiyama et al. [29] used CNNs to detect malware by its behavior. The authors used Procmon[1] to log the malware sample's interactions with the operating system (files opened, registry key modification, network behavior, etc.), and trained an RNN with LSTM units. Using the trained values of their hidden layers, they constructed an 'image' and applied several convolutional layers to reduce feature representation. Their best results were AUC=0.96 for malware detection.

Kolosnjaji et al. [27] presented a hybrid DNN architecture for malware classification by applying a CNN before an RNN. The convolutional layers were used to create n-gram representations of system call sequences. These 'n-grams' were fed into an RNN with LSTM units to classify the malware. Their model obtained 89% accuracy.

Yousefi-Azar et al. [30] and David et al. [31] presented an unsupervised method for PE malware classification. Using a deep autoencoder model, the studies showed that it is possible to classify malware by training a bottleneck AE to reduce the feature dimensions of each sample, thus creating a small representational vector. Yousefi-Azar et al. and David et al. achieved accuracy of 83% and 95% respectively.

Table 1 summarizes the related studies and shows the length of system call sequences used in each paper, along with its year of publication. The last row in the table indicates that the average system calls sequence length used in prior research ranged from 200 to 250, and we refer to this as the baseline sequence length in our initial experiments.

Study	***Year***	***Length of System-Calls Sequence Used***
Tobiyama et al. [29]	2015	350
Canzanese et al. [23]	2015	250-1500
Pascanu et al. [28]	2015	100
Kolosnjaji et al. [27]	2016	100
Yousefi-Azar et al. [30]	2017	150
Average		***200-250***

Table 1 - Length of system call sequences used in related studies that explored malware detection and classification

To the best of our knowledge, no study proposed and explored the use of sequences of system calls with attention mechanisms or transformer architectures for malware detection or classification. Also, as seen in the table, the longest sequence of system calls used in related work was 1,500. In this study we address these two gaps for the task of malware classification. If we are successful, such a method could also be used for the task of malware detection.

4. Research Questions

In this paper, we compare five architectures for malware classification using sequences of system calls: 1) LSTM, 2) LSTM with an attention layer, 3) bidirectional LSTM, 4) bidirectional LSTM with an attention layer, and 5) transformer. Our study is aimed at answering the following four research questions:

1. Does the use of attention and transformer architecture improve the ability to accurately classify unseen malware compared to state-of-the-art methods (based on the average sequence length of 200 used in previous work)?
2. Does the sequence length affect the accuracy of each of the classification architectures examined?
3. Does the sequence length affect the time complexity of each of the classification architectures examined?
4. What is the most cost-effective combination of architecture and sequence length that provides the best tradeoff between malware classification accuracy and time complexity (both in the training and test phases)?

5. Methodology

a) Data Collection

Our research focuses on portable executable (PE) malware aimed at the Windows 7 operating system, which is still widely used by many organizations and individuals. The malware collection we use is the academic repository provided by VirusTotal (VT) which consists of about 142,000 PE malware samples for Windows, collected between 2015 and July 2018. For each malware sample, we obtain the detection and malware type from over 60 antiviruses, using a JSON file provided by VT in the repository. The type of each malware (ground truth) is identified based on the majority malware type suggested among the top performing seven AVs[2] (as reflected by their detection rate for the entire repository). We remove the samples that were not detected by

[1] https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

[2] Per the agreement with VirusTotal, we cannot include any comparison between the AVs; thus no AVs will be named in this paper.

all of the seven AVs, leaving us with ~12,000 samples, which we classify into eight ground truth types using the abovementioned majority voting procedure between AVs. Our final dataset collection and its distribution can be seen in Figure 1.

b) Dynamic Malware Analysis and System Call Acquisition

Each malware sample is executed in **Cuckoo sandbox**, an open-source platform for dynamic malware analysis. The analyzed samples are given limited access to the Internet using a dedicated Ethernet port to prevent infection of our internal network. For each malware sample analyzed, Cuckoo sandbox produces a JSON file report containing various dynamic and temporal information that documents the behavior of the malware while it was executed. From the JSON report, we extract the sequence of observed system calls and arrange them according to their timestamp, irrespective of the specific thread that executed each system call. The reason behind mixing the threads is that malware might try and evade security mechanisms by splitting its functionality between threads, so that each thread executes a different part of the attack. Mixing and ordering the executed system calls based on their timestamp provides us with a comprehensive picture of malware sample's behavior. Our research assumes that the malware sample is executed successfully in the analysis environment and that the list of executed system calls is extracted accurately. Since very short sequences are not informative enough and might suggest that a problem occurred during the analysis or that the malware was not executed properly, we exclude reports with less than 200 system calls; thus, our final dataset collection consists of 10,974 malware samples distributed among eight malware types (Figure 1).

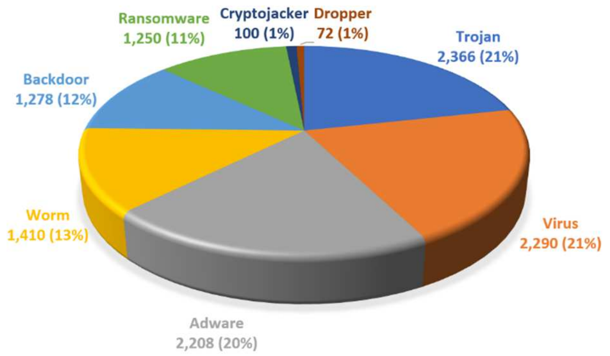

Figure 1 - Distribution of the malware samples in our data collection based on their malware type.

The analysis of our data collection described above results in a set of 293 distinct system calls. Before feeding the system calls to our neural network model, we map each system call to a numeric number. Figure 2 illustrates the sequence of system calls extracted from three variants of the 'Adload' adware. Each pixel represents a single captured system call mapped to a shade of gray, and each line contains 50 system calls. As can be seen in the figure, all three variants have the same behavior during the first ~4,000 system calls (first 75 lines), and later their behavior diverges. Our hypothesis is that attention and transformer architecture can automatically learn subsequences shared between variants and classify them as the same (correct) malware type and by doing so, improve the malware classification capabilities.

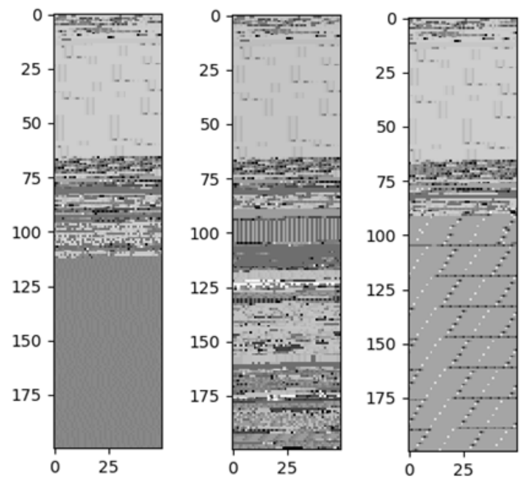

Figure 2 - Visualization of system call sequences for three 'Adload' (adware) variants.

b) System Call Sequence Preprocessing

Each malware sample in our repository invoked between three and 600,000 system calls. In the first preprocessing step, we translate each system call into a numeric value (between one and 293). Since most sequences are very long, they had to be broken down into subsequences; we thus divide them into subsequences of length L and use zero padding to ensure that all of the subsequences are the same length. For any given sequence of system calls S, which originated from a malware sample of class C, we divide S into non-overlapping subsequences of length L and label each subsequence as class C.

c) Proposed Malware Classification Methods

Before describing the baselines (state of the art methods) and the proposed learning methods, we provide an explanation of the training and classification phases which are the same for each method. During training, each subsequence S is fed to the model, and the model's weights are optimized to predict the correct class C, as suggested by [32]. To classify an unseen malware sample from the test set, we split the malware's system call sequence into subsequences of the same length L. We then feed each subsequence to the trained model to obtain a class prediction for that subsequence. The malware sample is then classified based on a majority vote between all the predictions.

For example, let us assume that we have a virus sample which invoked 3,900 system calls in our Cuckoo sandbox environment, and we want to use a classification model with a sequence length of 200. We divide the 3,900 observed system calls into 20 subsequences of 200 system calls each; the last subsequence contains 100 system calls and 100 zeros. For training, we add the 20 subsequences to our dataset, each one with the 'Virus' label. For classification, each subsequence is fed to the trained model, and we receive a predicted malware type for each. A majority vote based on all of the predictions provides our model's final prediction.

The first baseline model [32] consists of an LSTM layer followed by a dense layer with a softmax activation function and eight outputs, one for each malware type. We use the Adam optimizer ($lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.998$) and categorical cross-entropy as a loss function. This design is based on a recent study [32]. To prevent the model from overfitting to the training set, we keep 25% of the training set aside for validation and use an early stopping mechanism (stop training if the validation loss has not decreased for three consecutive epochs). The second baseline model is created by replacing the LSTM layer with bidirectional LSTM [32] - the motivation for that was to evaluate the effect on performance when analyzing the sequence in both directions. All models were implemented using Python and Keras library.

On top of each of the abovementioned baselines, we integrate an attention mechanism and implemented it, as was suggested in the original study by Bahdanau et al. [5]. Figure 3 presents an illustration of our LSTM network with an attention mechanism and demonstrates how it processes a sequence of system calls (s_0, s_1, s_2, s_3). A description of this process follows: The first system call s_0 is fed into the LSTM unit, and the output $\widehat{s_0}$ is fed into the same LSTM unit, along with the next system call, s_1. This process continues until all of the sequences have been processed; note that all the outputs of the LSTM units are saved so they can be fed into the attention layer, giving it a comprehensive view of the entire history and context of each system call in the sequence, thus reducing the diminishing gradient effects. The output of the attention layer is then fed into a dense layer with softmax activation to produce a classification prediction.

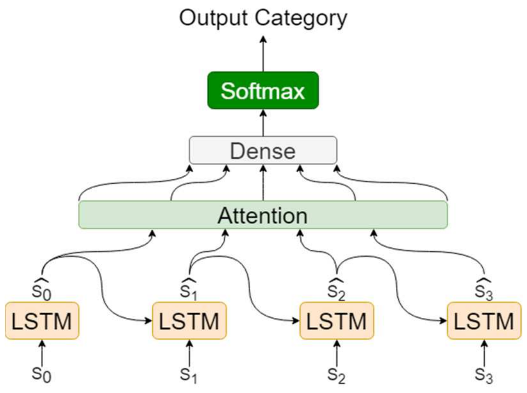

Figure 3 - LSTM and attention layer for malware classification

Since RNNs analyze a sequence from beginning to end, each item is analyzed in the context of previous items. However, for many sequence analysis tasks, it is useful to process the sequence in both directions simultaneously. Bidirectional LSTM [33] was introduced to perform this task. By having a view of future items, bidirectional LSTM can extract more context per item in the sequence.

Thus, we created our first proposed malware classification method which we believe will better capture, represent, and classify the malware behavior based on the system call sequences invoked by the malware sample.

Given the advantages of the transformer architecture over attention mechanisms, we also propose integrating a transformer(see Figure 4), based on the original implementation suggested by Vaswani et al.[6], as a third classification method in order to explore its potential in improving accuracy in the malware classification task.

Instead of relying on RNNs for sequence mining and analysis, the transformer uses a **multi-head attention** block which takes each system call in the sequence and maps it to three weight vectors: key, value, and query. To process an item in the sequence, we iteratively feed the item's query vector to the attention mechanism with the key and value vectors of each of the other system calls in the sequence. The attention layer performs dot product multiplication between the vectors, which produces an output vector for that 'head.' The last action is a concatenation of the output of all 'heads' which is fed to a linear layer. During training, the weights of the key/value/query vectors are optimized for each system call in the sequence for the task at hand. Figure 4 presents a transformer with a three-head attention block for the task of sequence classification.

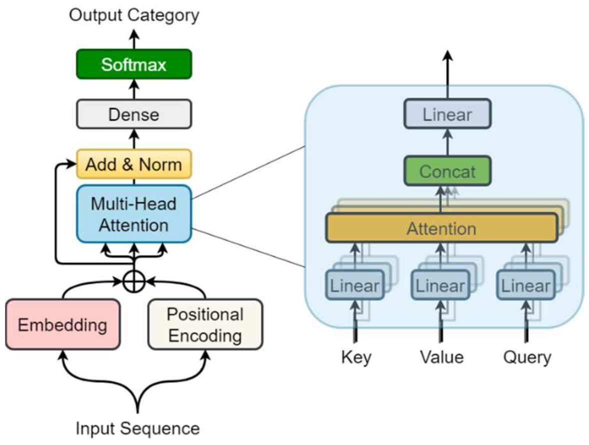

Figure 4 - Illustration of the transformer architecture for sequence classification

In total, our evaluation included five different malware classification architectures, two of which are baselines and three of which are architectures based on our proposed use of attention and transformer mechanisms.

6. Experimental Design

Before describing each of the three experiments performed, we explain the general procedure employed to train and evaluate the five malware classification models.

As explained above, the execution of each malware sample produced a sequence of system calls, which was divided into S non-overlapping subsequences of length L, thus each malware sample was represented by S subsequences, which were labeled with the same malware category.

We randomly chose eighty percent of the malware samples from each class, along with their system call subsequences, and used them to train our models. The remaining 20%, along with their system call subsequences, were left for testing. We repeated the splitting in and used the standard 10-fold cross-validation setup. The following experiments aimed at providing concrete answers to research questions presented in Section 4 were performed.

a) Experiment 1 – Accuracy Over Average Sequence Length

The first experiment is designed to evaluate the five classification methods' performance for the task of malware classification using sequence length L=200. This length was chosen based on the summary of the related work section presented in Table 1. We trained the five architectures using 10-fold cross-validation (50 models in total), with a sequence length of 200. For evaluation purposes, we calculated the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) rates of the test set for each model and calculated the accuracy of the model using equation (1):

$$(1)\ Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

The results of all folds were averaged and are presented in the Results section.

b) Experiment 2 – Accuracy vs. Sequence Length

In this experiment, we explore the impact of the sequence length on the accuracy. Such an experiment can shed light on the pros and cons of each architecture and enable us to identify the best configuration (model and sequence length for the malware classification task). To evaluate the effect of different sequence lengths on our models, we trained the five architectures with 22 different sequence lengths (10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1,000, 1,500, 2,000, and 4,000), for a total of 1,100 trained models. The same training and test procedure used in experiment 1 was employed.

c) Experiment 3 – Time Complexity vs. Sequence Length

Out last experiment is designed to evaluate the impact of the sequence length on the time complexity. Such an experiment can help us better understand the cost, in terms of the time required for both the training and test phases, for the purpose of future application of the proposed methods. Since we used an early stopping mechanism, each model was trained for a different number of epochs. To compare the time complexity between different models, we calculated the average epoch time in the training phase (in seconds); for the test phase we measured the time it takes for each model to provide a classification prediction.

7. Results

a) Experiment 1 – Accuracy Over Average Sequence Length

As can be seen in Figure 5, the results show that for sequence length *L*=200, the addition of an attention layer to the baseline LSTM and bidirectional LSTM models improved the results slightly. In each case, adding an attention layer to the baseline improved the average accuracy by 0.9% (from 92.2% to 93.1% for the baseline LSTM and from 93.0% to 93.9% for the baseline bidirectional LSTM). In contrast, the transformer model achieved significantly lower accuracy (an average of 86.6%) with our dataset.

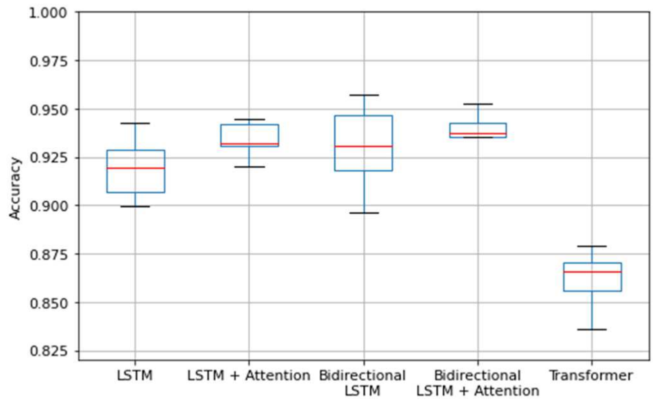

Figure 5 - The accuracy boxplot for the five different classification architectures with sequence length L=200.

b) Experiment 2 – Accuracy vs. Sequence Length

As can be seen in Figure 6 (note the log scale on the x-axis), the results show that adding an attention layer for sequence lengths shorter than 400 system calls slightly improves the accuracy compared to the baseline models (up to 2.8%), a trend that is consistent with the results of experiment 1. However, for longer sequence lengths, we can observe that the attention mechanism improves the accuracy by a more significant rate (up to 6%). More importantly, beyond a sequence length of 400 the accuracy starts to drop for all four RNN variants, while the transformer architecture continues to improve significantly, until it reaches 94.7% accuracy for sequence length *L*=4000. This result is on par with the accuracy of our top-performing RNN across all of our experiments - a bidirectional LSTM with attention and *L*=80, which achieved 94.8% accuracy.

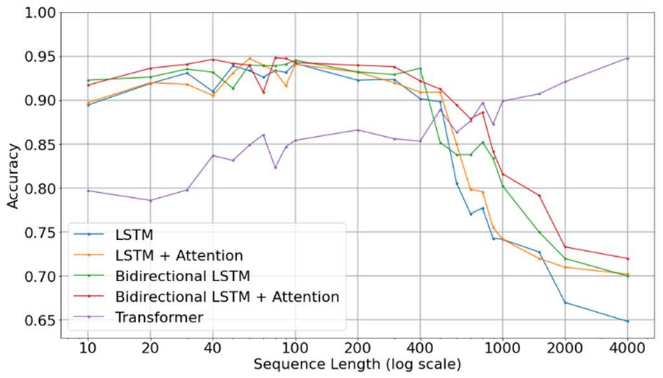

Figure 6 - The accuracy for the five different models with various sequence lengths.

c) Experiment 3 – Time Complexity vs. Sequence Length

From a performance point of view, the training time increases when adding attention layer to the baseline models (as can be seen in Figure 7). On our dataset, the transformer is consistently and significantly superior compared to the RNNs in terms of training time for all sequence lengths examined.

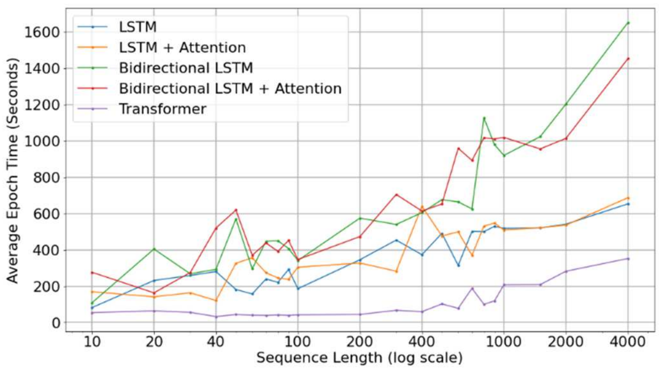

Figure 7 – Average training time per epoch for the different models and sequence lengths.

When considering which model is suited for applicative use when the time it takes to provide a classification (the prediction phase) plays a more significant role, it is important to compare the time performance of models after the training phase is over. Thus, we also provide a comparison of the average time it takes to classify a given malware (Figure 8), where is it easy to see that the transformer model is the fastest.

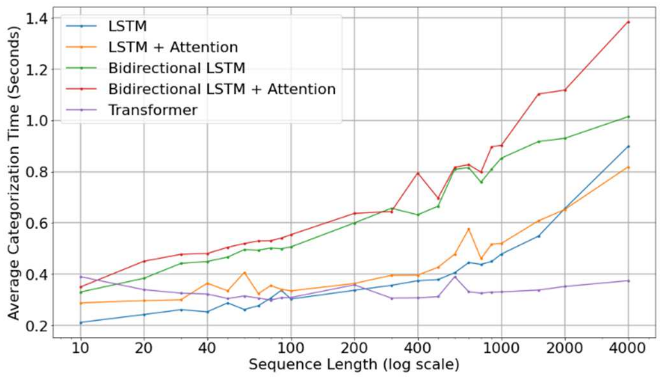

Figure 8 - Average malware classification time per malware sample.

d) Cost-Effectiveness: Accuracy vs. Time Complexity

To answer our fourth research question, we need to assess the cost-effectiveness of each architecture and sequence length combination. Thus, we devised the following metric (2):

$$(2)\ Cost\text{-}Effectiveness_{aL} = \frac{Acc_{aL} - Acc_{LSTM10}}{T_{aL}/T_{LSTM10}}$$

Acc_{aL} – accuracy of architecture *a* with sequence length *L*.

Acc_{a10} – accuracy of baseline LSTM with sequence length 10.

T_{aL} – average training/classification time for architecture *a* with sequence length *L*.

T_{LSTM10} – average training/classification time for the baseline LSTM with sequence length 10.

The baseline for this metric is based on the results we obtained for LSTM with sequence length *L*=10. This proposed metric divides the improvement (in terms of additional accuracy) by the factor of added time complexity. Higher positive values of this metric indicate better accuracy with shorter training/classification time compared to the baseline. On the other hand, lower negative values indicate that the model accuracy decreases as the time complexity grows. When the accuracy for sequence length *L* (Acc_{aL}) is equal to the baseline accuracy (Acc_{LSTM10}), the cost-effectiveness is zero. We applied the cost-effective metric on our results for training time complexity (Figure 9) and classification time complexity (Figure 10). As these results show, the most cost-effective combination of architecture and sequence length (on average) for both training and classifying varies depending on the sequence length; for sequences shorter than 800 system calls the most cost-effective is the baseline LSTM, while for sequences longer than 800 the transformer dominated.

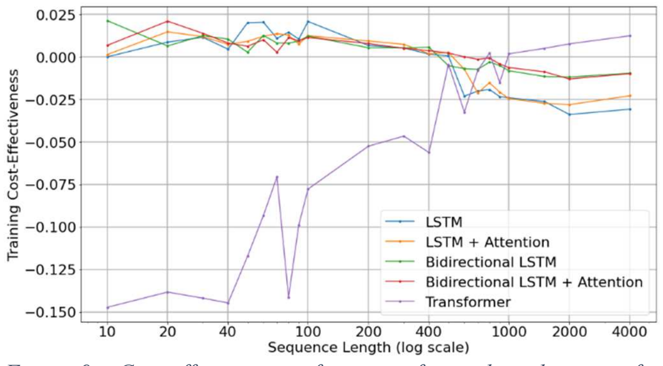

Figure 9 - Cost-effectiveness of training for each architecture for different sequence lengths

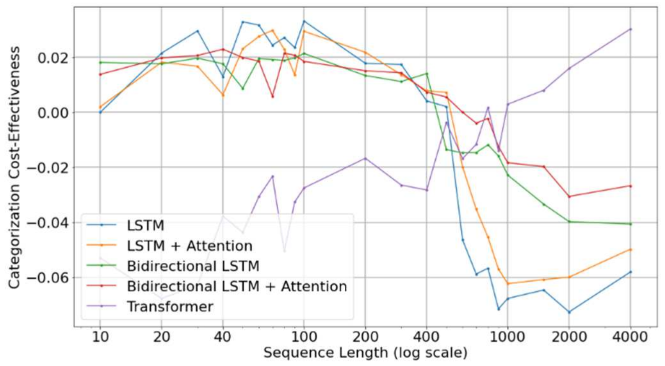

Figure 10 - Cost-effectiveness of classification for each architecture for different sequence lengths

8. Discussion & Conclusions

The first conclusion from our experiments is that when feeding longer sequences to our models, more complex context is extracted for the task of malware classification. While any information that exists in a short sequence also exists in a longer sequence, longer sequences contain wider context, which improves the malware classification accuracy and can better capture the subtle difference that exist between malware families, based on the order of the system calls they invoke during their execution.

The decrease in accuracy observed when using sequences of more than 400 system calls demonstrates the diminishing gradient effect. The LSTM units slowly 'forget' information as they process new items, making it harder to correlate between items that are far away from each other in the sequence. This is diminished when an attention layer is added to the LSTM. The use of an attention layer improves the accuracy of LSTM for almost all sequence lengths over 400. This demonstrates the attention mechanism's ability to correlate between system calls that are very far from one another in the sequence, supporting our hypothesis.

The transformer model achieves better accuracy than the LSTM and attention variants for sequence lengths of 800 system calls or more. While the transformer's accuracy is low for very short sequences, it improves steadily when longer sequences are used. Eventually the transformer architecture is able to achieve the same results as the best RNN model.

The cost-effectiveness analysis we performed shows that the transformer architecture is inferior for short sequences in comparison to the RNN variants. However, since it steadily improves in terms of cost-effectiveness (unlike the RNN variants) for very long sequences (1,000 system calls and over), the transformer architecture is superior in this aspect. This is a strong indication that the transformer architecture can serve as the basis of a new system call analysis method for the task of malware classification.

9. Future work

The first direction planned for future work will focus on extracting system calls from malware designed for different operating systems (Windows 10, Linux, and macOS). Although Windows 7 is still being widely used by many organizations, Microsoft has replaced it with Windows 10, with new system calls and new kernel functionality. Executing the malware in our repository with the Windows 10 operating system will likely result in a different sequences of system calls, which makes the models trained in the current research irrelevant for Windows 10. In addition, while our research focused on malware classification, we believe that a model based on our architecture will also perform well for malware detection (while including also benign executable samples in our experiments). Such detection models could then be incorporated into dynamic analysis tools and improve detection capabilities and could even be used in near real time to detect malware as early as possible.

10. References

[1] A. Cohen and N. Nissim, "Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory," Expert Syst. Appl., vol. 102, pp. 158–178, 2018.

[2] D. Nahmias, A. Cohen, N. Nissim, and Y. Elovici, "Deep feature transfer learning for trusted and automated malware signature generation in private cloud environments," Neural Networks, vol. 124, pp. 243–257, Apr. 2020.

[3] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era - A state of the art survey," ACM Comput. Surv., vol. 52, no. 5, 2019.

[4] T. Bell, "The concept of dynamic analysis," ACM SIGSOFT Softw. Eng. Notes, vol. 24, no. 6, pp. 216–234, 1999, doi: 10.1145/318774.318944.

[5] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," pp. 1–15, 2014.

[6] A. Vaswani et al., "Attention Is All You Need," no. Nips, 2017, doi: 10.1017/S0140525X16001837.

[7] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," Neural Comput., 2000, doi: 10.1162/089976600300015015.

[8] J. Dike, "User-mode linux," 2001.

[9] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," Nature. 2015, doi: 10.1038/nature14539.

[10] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, no. 8, pp. 1798–1828, Aug. 2013, doi: 10.1109/TPAMI.2013.50.

[11] A. L. Caterini and D. E. Chang, "Recurrent neural networks," in SpringerBriefs in Computer Science, 2018.

[12] J. Schmidhuber, "Deep Learning in neural networks: An overview," Neural Networks. 2015, doi: 10.1016/j.neunet.2014.09.003.

[13] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," IEEE Trans. Signal Process., 1997, doi: 10.1109/78.650093.

[14] I. Sutskever, J. Martens, and G. Hinton, "Generating text with recurrent neural networks," 2011.

[15] K. Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," J. Biol. Chem., vol. 281, no. 49, pp. 37275–37281, Jun. 2014, doi: 10.1074/jbc.M608066200.

[16] S. Hochreiter and T. U. Munchen, "the Vanishing Gradient Problem During Learning," Int. J. Uncertainty, Fuzziness Knowledge-Based Syst., vol. 2, pp. 107–116, 1998, [Online]. Available: http://www.bioinf.jku.at/publications/older/2304.pdf.

[17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," IEEE Int. Conf. Rehabil. Robot., vol. 2015-Septe, pp. 119–124, Dec. 2014.

[18] Y. Wang, M. Huang, L. Zhao, and X. Zhu, "Attention-based LSTM for aspect-level sentiment classification," 2016, doi: 10.18653/v1/d16-1058.

[19] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo, "Image captioning with semantic attention," 2016.

[20] K. Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," 2017.

[21] T. J. O'Shea, L. Pemula, D. Batra, and T. C. Clancy, "Radio transformer networks: Attention models for learning to synchronize in wireless systems," 2017.

[22] H. Liu, J. Lu, J. Feng, and J. Zhou, "Two-Stream Transformer Networks for Video-Based Face Alignment," IEEE Trans. Pattern Anal. Mach. Intell., 2018, doi: 10.1109/TPAMI.2017.2734779.

[23] R. Canzanese, S. Mancoridis, and M. Kam, "System Call-Based Detection of Malicious Processes," Proc. - 2015 IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2015, pp. 119–124, 2015, doi: 10.1109/QRS.2015.26.

[24] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344), 1999, pp. 133–145, doi: 10.1109/SECPRI.1999.766910.

[25] C. Lim, J. T. Juwono, C. Lim, and A. Erwin, "A Comparative Study of Behavior Analysis Sandboxes in Malware Detection," no. November 2015, 2016.

[26] R. Mosli, R. Li, B. Yuan, and Y. Pan, "Automated malware detection using artifacts in forensic memory images," 2016 IEEE Symp. Technol. Homel. Secur. HST 2016, 2016, doi: 10.1109/THS.2016.7568881.

[27] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9992 LNAI, pp. 137–149, 2016

[28] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Apr. 2015, pp. 1916–1920, doi: 10.1109/ICASSP.2015.7178304.

[29] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware Detection with Deep Neural Network Using Process Behavior," 2016 IEEE 40th Annu. Comput. Softw. Appl. Conf., pp. 577–582, 2016.

[30] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," Proc. Int. Jt. Conf. Neural Networks, vol. 2017-May, pp. 3854–3861, 2017.

[31] O. E. David and N. S. Netanyahu, "DeepSign: Deep learning for automatic malware signature generation and classification," in 2015 International Joint Conference on Neural Networks (IJCNN), Jul. 2015, pp. 1–8.

[32] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers, vol. 7462. Springer International Publishing, 2012.

[33] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," in Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., vol. 4, pp. 2047–2052.