Modeling Malware as a Language

Yara Awad
Department of Computer Science
Boston University
Boston, MA, USA
Email: awadyn@bu.edu

Mohamed Nassar and Haidar Safa
Department of Computer Science
American University of Beirut
Beirut, Lebanon
Email: [mn115, hs33]@aub.edu.lb

***Abstract*—Malware detection and malware construction are evolving in parallel. As malware authors incorporate evasive techniques into malware construction, antivirus software developers incorporate new static and dynamic analysis techniques into malware detection and classification with the aim of thwarting such evasive techniques. In this paper, we propose a new approach to static malware analysis, aiming to treat malware analysis as natural language analysis. We propose modeling malware as a language and assess the feasibility of finding semantics in instances of that language. We concretize this abstract problem into a classification task. Given a large dataset of malware instances categorized into 9 classes, we isolate strong semantic similarities between malware instances of the same class and classify unknown instances by strength of similarity to a class.**

Our approach consists of a proposed method for defining a malware-language, where malware instances are documents written in that language. We use the word2vec model to generate a computational representation of such documents and choose a document-distance as the measure of semantic closeness between them. We classify malware-documents by applying the k nearest neighbors algorithm (kNN). Validating our model using leave-one-out cross validation, we record a classification accuracy of up to 98%. We conclude that we can find, and ultimately manipulate semantics in malware.

***Index Terms*—malware classification, static analysis, natural language processing.**

I. Introduction

Malware detection and classification originally comprised of signature-based and heuristic-based analysis methods [2]. Signature-based methods characterize malware through finding recognizable strings in its binary. Heuristic-based techniques characterize it through finding recognizable behavioral patterns across its execution. Besides being prone to evasion, these methods only improve after an unknown malware instance infects a system and becomes available for analysis.

Alternatively, many approaches have proposed malware classification through machine learning [1] [2]. Essentially, all approaches comprise a feature extraction component and a classification component. The classification component is trained on features extracted from malware executables. Some approaches focus on feature extraction through static analysis [6]-[11] while others focus on feature extraction through dynamic analysis [13]-[17] Figure 1 provides a step-wise illustration of static and dynamic analysis approaches for classifying a malware executable using machine learning techniques.

Static analysis techniques rely on the presence of syntactic properties in malware executables for feature extraction.

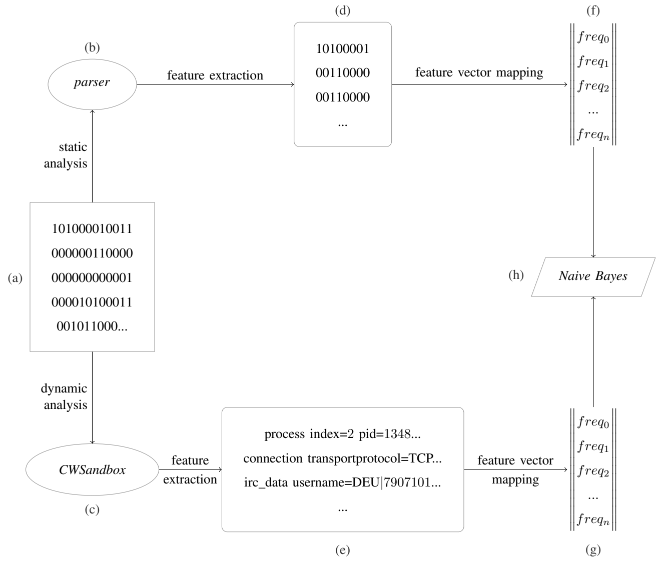


Fig. 1: (a) Unknown malware executable. (b) Static analysis of the malware executable through parsing its binary. (c) Dynamic analysis of the malware executable through running it in a sandbox environment, CWSandbox [12]. (d) Statically extracted byte n-grams. (e) Dynamically extracted CWSandbox behavioral report containing the performed API calls. (f) Vector of frequencies of the byte n-grams in the malware executable. (g) Vector of frequencies of the API calls in the behavioral report. (h) Naive Bayes machine learning classifier.

Though they are widely used, it is common for attackers to evade them. Evasion techniques such as code obfuscation, data obfuscation, the use of opaque constants and predicates, and the use of polymorphic code disrupt the syntax of malware, making it difficult to classify an unknown malware instance syntactically [3]. As a result, dynamic analysis techniques have been used apart from, or along with, static analysis techniques in order to achieve better malware detection and classification results.

Dynamic analysis techniques attempt to extract semantic properties from malware executables. This is achieved by running a malware executable in some secure environment while its execution behavior is monitored at runtime. Dynamic analysis characterizes malware behavior by characterizing the changes that the corresponding malware instance incurs upon a system.

Even though dynamic analysis overcomes evasive techniques that hinder static analysis, it may still be of little use when exposed to more powerful evasion [4]. Malware that recognizes the environment in which it runs can recognize its presence inside a sandbox or a virtual environment and can alter its execution accordingly in order to evade detection. Moreover, malware that runs in kernel-space need not use system calls to perform its tasks, and hence it can neither be detected nor classified through system call analysis. Such malware instances are difficult to analyize dynamically.

In this paper, we present a malware classification method in which static analysis serves a similar purpose to semantic-based dynamic analysis. We suggest modeling malware as a language. A malware executable would be represented as a document-like instance of that language. The damage that a malware instance will incur on a system would be implicitly stated through its document representation. Consequently, malware instances whose document representations are close in document-distance should belong to the same class of malware.

Section III explains our approach, which comprises of statically transforming malware executables to document-like instances, modelling these instances using the word2vec model [19], and measuring the document-distances between them using the word mover's distance (WMD) algorithm [21]. Using document-distances as the classification measure in k-nearest-neighbors (kNN) classification, we collect malware classification results of up to 98% accuracy, which we report in section IV. We attribute the accuracy of our classification to the notion that there is value in extracting semantic meaning from malware executables. We believe that this notion may be extended to incorporate finding semantic meaning in any executable. This would allow us to not only classify malware, but to detect it among non-malware instances, using an approach derivative from the one we take here.

II. Related Work

Static malware analysis involves the extraction of features from malware executables before their execution begins. In an early work, Schultz et al. extracted strings from text sequences in executable binaries [6]. They trained a Naive Bayes classifier to predict incoming executables as malware or benign, based on the frequencies of certain strings in the executables. In another attempt, the authors used features related to the system resources that an executable is destined to access, such as dynamically linked libraries and their constituent functions. They classified executables by the presence or absence of some of these features. Another work, proposed by Nataraj et al., treated the entire malware binary as an image vector [10] and used the euclidean distance between malware image vectors to classify incoming malware executables. A slightly different approach, proposed by Kong and Yan, classifies malware instances through their attributed function call graphs [11].

Often, proposed approaches to malware classification treat the analysis of malware as that of text. Many of them apply feature extraction to collect byte n-grams and opcode n-grams from executable binaries, such that an extracted n-gram term compares to a word entity. Feature vectors of n-grams are hence treated as feature vectors of words. Schulz et al. used such byte n-grams to train a Multinomial Naive Bayes classifier, which they used to classify malicious and benign executables [6]. Kolter and Maloof used them to train an ensemble of classifiers [7], from which the J48 Decision Tree produced the best classification.

Moskovitch et al. extracted byte n-grams from executable binaries and performed feature selection to isolate the most significant n-gram terms [8]. They treated each executable as a text document and mapped its constituent n-grams to a term-frequency vector which they used as a feature vector in their classification model. In another work, the authors disassembled executable binaries in order to facilitate the extraction of opcode n-grams from the disassembled files [9]. Their aim was to train their predictive classifiers to recognize common 'engines', or particular sequences of assembly instructions, in malware code. They chose to extract opcodes and disregard opcode parameters in an attempt to bypass evasive techniques that obfuscate memory addresses and values.

Dynamic malware analysis involves monitoring malware behavior at runtime, inside some safe environment, in order to extract behavioral features from execution patterns. Features extracted dynamically tend to retain some semantic context from a malware instance. In a work by Biley et al., the authors collected execution traces from running malware in a virtual machine and mapped these traces to feature vectors of state changes [16]. They performed hierarchical clustering on these vectors, using normalized compression distance (NCD) as a similarity measure. Lee et al. propose a similar workflow. Upon observing malware interactions with system resources, the authors built behavioral profiles which they used to train a k-medoid clustering algorithm [17].

Analysis of system calls is another common dynamic analysis technique used to facilitate malware classification. Using inline function hooking, Reick et al. collected behavioral reports of the system calls made by a malware instance as it executes in a sandbox environment [13]. For each malware instance, they compiled a feature vector of system call frequencies. In a related work, the authors modeled the system calls numerically as 'malware instructions' (MIST) and produced behavioral profiles of MIST sequences for all malware instances [14]. They then extracted MIST q-grams and used them to compile feature vectors of MIST q-gram frequencies.

In this project, we build on these related methodologies through presenting a new approach to static malware analysis. We choose not to extract information from malware execution traces or from malware source code. Instead, we propose modeling the malware source code as a language through employing word2vec embedding techniques.

III. Methodology

Our malware classification approach hinges on modeling malware as a language. We treat a malware instance as text and assume that it belongs to a language. Language processing techniques, such as word2vec, recurrent neural networks, and long-short-term memory networks may be used to facilitate this modeling. Such techniques work well for classification tasks that involve spoken languages. If we can design a good language model for malware, we may achieve similar good results for malware classification tasks.

Figure 2 shows a high level overview of our approach. We begin by designing a sample *malware-language* with concepts of a vocabulary, words, and documents. To that end, we define a *malware-vocabulary*, a *malware-word*, and a *malware-document*. We then transform the assembly code of malware instances to a form that conforms to our proposed malware-language. Every malware instance is transformed to a malware-document of malware-words, as illustrated in figure 2(a). Next, we model the transformed malware instances using the word2vec model [19] [20]. By doing so, we are able to represent malware-documents and malware-words as mathematical objects, such that we may perform computations on them. Every malware-word is modeled as an embedding vector. The embedding vector of a word is a numerical representation of the semantic context within which the word is most likely to occur. A malware-document, in turn, is modeled as a matrix of its constituent malware-word embedding vectors, as seen in figure 2(b). We chose *document-distance* as the measure of closeness between malware-documents and compute it using the word mover's distance algorithm (WMD) [21]. Figure 2(c) shows the final step of our approach, in which we use the computed distances as the classification measure by which we perform our *kNN* classification.

A. Malware-Language Design

A malware-language design assigns definitions to the following terms:

- A *malware-document* is a set of malware-words.
- A *malware-word* is the smallest unit of the language. It represents a single assembly instruction in the malware executable file. A malware-word can be a full instruction or some abstraction of it. That is, we may consider an instruction opcode alone as a malware-word (e.g. *push, mov, jmp, ...*), or we may consider an instruction opcode with its operand types as a malware-word (e.g. *mov register memory*).
- A *malware-vocabulary* is the set of all possible malware-words in the malware-language.

Figure 3 shows the stages of our malware-language design process. We present two designs. For the first language design, we define a malware-word to be any x86-64 opcode, as shown in figure 3(a). For the second design, we define a malware-word to be the concatenation of the x86-64 opcode of an assembly instruction and abstractions of the instruction's operands. In particular, we transform constants, addresses, offset values, and general purpose register names in instruction operands into generic representations. We also abstract out function names and addresses. Figure 3(b) shows a sequence of malware-words that are the result of such a transformation.

We build the malware-vocabulary from the set of all malware-words and define a *malware-document* as some sequence of malware-words. It is important to note that the malware-vocabulary of our second malware-language design is significantly larger than that produced by the first language design. A larger vocabulary provides us with a more comprehensive list of malware-words which we may use to isolate differences in dissimilar malware-documents.

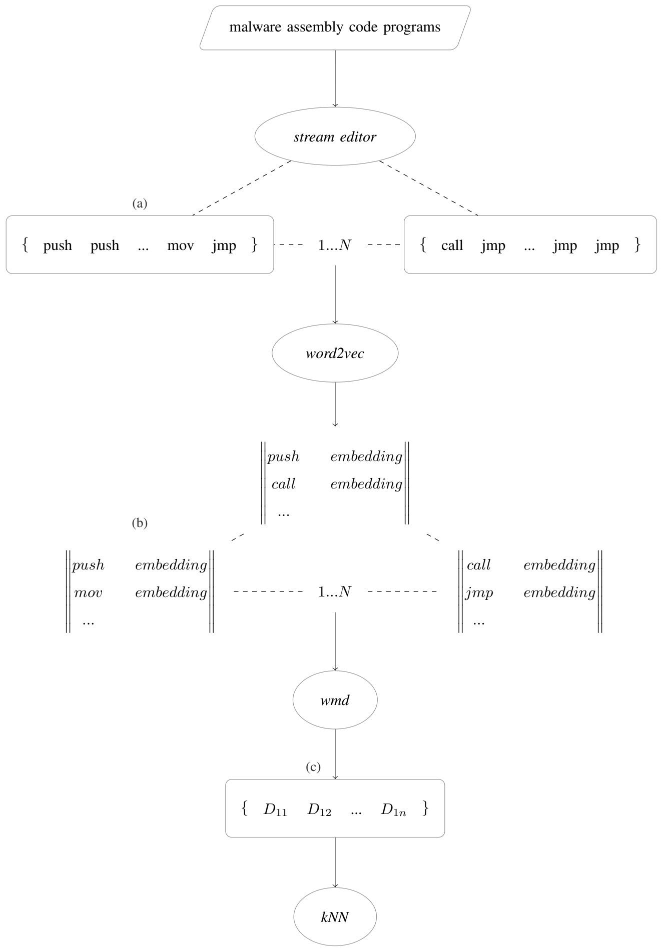
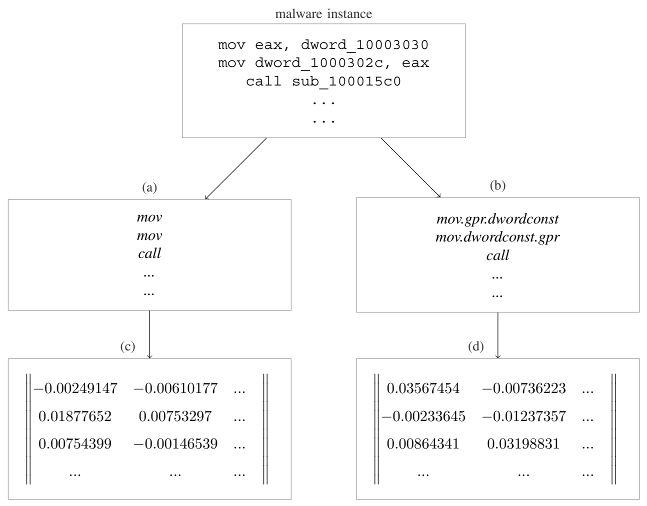


Fig. 2: (a) malware-documents of malware-words (b) malware-document matrices of malware-word embeddings (c) document-distances between some unknown malware instance and N known malware instances

B. Classification within a Malware-Language

We classify a malware instance by classifying its malware-document representation. We use a *kNN*-based classifier in our experiments, but our approach can be easily extended to accommodate other classifiers. We choose *document-distance* between malware-documents to be the measure by which we find a malware-document's nearest neighbors and, consequently, classify it.

Algorithm 1 illustrates the steps taken for malware classification. We construct a set of known (labeled) and unknown malware-documents. Given an unknown malware-document, we compute the distances between it and some set of known malware-documents. We estimate that the unknown malware-document would belong to the class of its closest neighbor in



```
mov eax, dword_10003030
mov dword_1000302c, eax
call sub_100015c0
...
...
```

Fig. 3: (a) malware-words of our first malware-language design (b) malware-words of our second malware-language design (c)(d) malware-documents as matrices of malware-word embedding vectors

Algorithm 1: Malware Classification by kNN

```
Data: $a$: unknown malware, $data$: set of known malware,
      $distances$: sorted dictionary of $(b, dist)$ pairs,
      where $b \in data$ and $dist$ = distance of $a$ from $b$
Result: $class$: classification result as a malware class
1 $closest \leftarrow []$;
2 for $i \in range(1, K)$ do
3     $close\_neighbor \leftarrow distances[i][b]$;
4     $close\_class \leftarrow class(close\_neighbor)$;
5     $closest.append(close\_class)$;
6 end
7 $class \leftarrow closest.most\_common()$;
```

distance, or to the class to which the majority of its closest neighbors belong.

C. Document-Distance Computation

We use the word mover's distance algorithm (WMD), proposed by Kusner et al. [21], to compute document-distances between malware-documents. WMD leverages advances in word and document vector-space models, making use of the word2vec model [19] [20].

1) Word2vec: Given a set of documents written in some language, word2vec maps the words in these documents to *m* dimensional *word embedding vectors*. These vectors characterize words by context: words that appear in similar semantic contexts within documents are mapped to vectors that are in close proximity within the *m* dimensional embedding space. Consequently, words with similar meanings cluster together in the embedding space. A malware-document may be visualized as a matrix of its constituent malware-word embedding vectors. Malware-document embedding matrices should, then, coexist within some *n-by-m* dimensional space, where *n* is the total number of malware-words in the vocabulary.

2) WMD: WMD computes the document-distance between two documents, A and B, as the cost of *transporting* all p embedded words of A to all q embedded words of B. The elementary cost of transporting word i to word j is defined by the Euclidean distance, $c(i, j)$, in the word2vec embedding space. The algorithm finds an optimal flow matrix, T, which it uses to compute the minimum overall transportation cost between the words of two documents. When word i is transported across this distance to word j, it is either transported in full or in part, depending on the value of T_{ij}. If words i and j are identical, i is transported fully to j. Otherwise, it is transported in part, such that, the fraction of word i that is transported to word j is proportional to the semantic similarity between the two words.

$$\begin{aligned}
\underset{T}{\text{minimize}} \quad & \sum_{i=1}^{p}\sum_{j=1}^{q} T_{ij}c(i,j) \\
\text{subject to:} \quad & T \geq 0, \\
& \forall i \in \{1,\ldots,p\} : \sum_{j=1}^{q} T_{ij} = d_A(i), \\
& \forall j \in \{1,\ldots,q\} : \sum_{i=1}^{p} T_{ij} = d_B(j) \\
\text{where} \quad & d_X(k) \text{ is the frequency of word } k \\
& \text{in document } X
\end{aligned}$$

IV. Experiments and Results

A. Dataset

We use a malware dataset provided by Microsoft to test our proposed malware classification method. It comprises of 10868

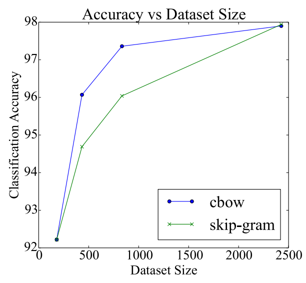


Fig. 4: Accuracy of malware classification, using **language design 1**

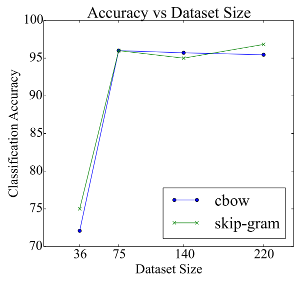


Fig. 5: Accuracy of malware classification, using **language design 2**

TABLE I: 2427 Malware-Documents from **Language Design 1**: Classification Accuracy

		Value of k for kNN						
		1	2	5	9	10	20	
word2vec window size	5	97.9%	98.06%	97.12%	96.33%	96.17%	94.77%	cbow
	7	97.49%	97.36%	96.91%	96.04%	96.84%	94.93%	
	10	97.53%	97.36%	96.66%	95.76%	95.39%	94.66%	
	5	97.94%	97.69%	96.87%	95.76%	95.51%	94.52%	skip gram
	7	97.82%	97.53%	96.5%	96.29%	96.04%	95.06%	
	10	97.53%	97.73%	96.66%	96.21%	95.88%	94.6%	

TABLE II: 140 Malware-Documents from **Language Design 2**: Classification Accuracy

		Value of k for kNN					
		1	2	5	9	10	
word2vec window size	5	95.71%	92.14%	89.30%	79.30%	80.71%	cbow
	7	95.00%	92.14%	89.30%	80.71%	80.71%	
	10	95.00%	92.14%	89.30%	80.71%	80.71%	
	5	95.00%	90.71%	92.86%	82.86%	81.43%	skip gram
	7	95.00%	90.71%	92.86%	81.43%	81.43%	
	10	95.00%	90.71%	92.14%	81.43%	81.43%	

TABLE III: 180 Malware-Documents from **Language Design 1**: Classification Accuracy

		Value of k for kNN					
		1	2	5	9	10	
word2vec window size	5	92.22%	83.33%	83.33%	79.44%	77.78%	cbow
	7	92.78%	90.56%	86.67%	81.67%	81.67%	
	10	89.44%	90.56%	82.22%	83.89%	83.89%	
	5	92.22%	87.22%	85.56%	77.78%	76.67%	skip gram
	7	90.56%	89.44%	82.22%	78.33%	78.33%	
	10	93.33%	94.44%	88.89%	80.00%	77.78%	

malware instances available as disassembled binaries. These instances belong to 9 classes: *Ramnit, Lollipop, Kelihos_ver3, Vundo, Simba, Tracur, Kelihos_ver1, Obfuscator.ACY, Gatak*. We use the *sed* stream editor to transform all disassembled malware executables in this dataset to malware-documents. Stream editing enables us to convert tokens of the malware assembly code to tokens on our designed language. The resulting malware-documents are compiled into a corpus of 10868 instances of a malware-language. We partition this corpus into several datasets of different sizes, which we use to train our model and test our classification.

B. Implementation

We use the Gensim Python library [22] for modeling and computation tasks. We use the Gensim implementation of word2vec to build our malware-language model from malware-word embeddings. We use the Gensim implementation of the WMD algorithm, wmdistance, to compute document distance. We validate our method using leave-one-out cross validation under several experimental setups. In these setups, we vary the dataset sizes to note the amount of training that our model needs. We also vary the word2vec parameters to study whether a subset of these parameters produces a better model than any other. We test against different values of k in kNN classification in order to better visualize our radius of similarity.

C. Results

Figures 4 and 5 show the accuracy curves obtained upon testing our classification against different dataset sizes from our first and second malware-language designs, respectively. For these tests, we fixed a word2vec window size of 5, as is the default choice in the Gensim word2vec implementation, and a value of 1 for k in kNN classification. The choice of k was meant to reflect how accurate we can be in finding the closest neighbor that is of the same class as a target unknown malware. We compared the two word2vec learning algorithm variants, namely cbow and skip-gram, which resulted in similar classification performance.

As expected, we observed that classification accuracy increased with larger datasets. More interestingly, with smaller datasets, we observed that classification of malware instances modeled with the second language design gave better results than did classification of malware modeled with the first language design. Moreover, results attributed to the second language show rapidly improving performance with small increases to the dataset sizes used. We attribute these observations to the richness in the vocabulary of the second language which captures more semantic properties of the malware-documents. Hence, we require less malware-documents from the second malware-language to isolate semantic differences between malware classes than we do from the first malware-language. However, this comes at the expense of the superior computational cost that the second malware-language design presents due to the larger vocabulary it creates.

Tables I, II, and III show malware classification accuracy results from several experimental setups. In these setups, we used fixed dataset sizes and varied the word2vec window sizes between 5, 7, and 10, the word2vec learning algorithms between cbow and skip-gram, and the values of k in kNN classification between 1, 2, 5, 9, and 10 k nearest neighbors. We observed no significant variance in classification accuracy upon using different word2vec parameters. However, we observed a notable decrease in accuracy as the value of k increased. This decrease may be explained by the notion that it becomes more likely for a dissimilar malware to be found among the k nearest neighbors of some unknown malware if k is large. We recorded the highest classification accuracy results for small values of k, particularly for $k = 1$ and $k = 2$. We recorded comparable accuracy results from classification using a large dataset of the first malware-language in Table I and a

TABLE IV: Confidence Measures

		Dataset Size			
		36	75	140	220
Value of k in kNN	1	100.00%	100.00%	100.00%	100.00%
	2	87.50%	93.33%	93.93%	96.14%
	5	53.33%	77.33%	86.71%	92.64%
	9	40.12%	63.70%	83.49%	87.22%
	10	37.22%	59.87%	82.00%	85.95%

small dataset of the second malware-language in Table II. This observation reinforces the idea that the second language design dominates in terms of classification accuracy.

We report average confidence levels of classification decisions made upon using datasets from our second malware-language model in table IV. The confidence level of a decision is the ratio of correct observations to the ratio of total observations. In our experiments, the confidence level of an unknown malware's classification is the ratio of nearest neighbors that belong to the decided malware class to nearest neighbors that do not. As k increases, confidence levels decrease, as it becomes more likely for dissimilar malware to coexist with similar malware around an unknown instance.

V. Conclusions and Future Work

Semantic-based analysis of malware has been implemented through the use of a variety of dynamic analysis techniques. In this paper, we proposed modeling malware as a language for the purpose of performing such semantic-based analysis. This technique allows us to extract sufficient semantic context from malware instances statically, reducing the complications associated with dynamic malware analysis. We designed a *malware-language* and succeeded in extracting semantic context from its *malware-documents* through using the word2vec model. We were able to use this semantic context, along with the word mover's distance algorithm, for kNN-based classification, through which we classified unknown malware instances into 9 malware classes with classification accuracy results reaching over 95%.

Our results show that language processing techniques used on natural languages may be used on malware for the purpose of malware detection and classification. Moreover, they give us insight that both benign and malware instances may be modeled as documents of some language. Such insight can help in the discovery of new malware families through detection of zero-day malware. As future work, we propose using sequence encoders, such as recurrent neural networks (RNN) and long-short term memory networks (LSTM), as alternative malware classifiers. It is possible that one limitation of the word2vec model is its disregard for word ordering. Sequence encoders, on the other hand, are dependent on the linearity of language. They extract context from an ordered sequence of words against time. This may be a powerful tool for detecting and classifying malware, particularly when sequences of malware instructions must occur in a certain order. This may also be helpful in evading obfuscation techniques that aim to crowd malware with code which has no particular role in the sequence of instructions that the malware must perform. Sequence encoders could be able to disregard such code while maintaining track of the main thread of malware instructions.

References

[1] E. Gandotra, D. Bansal, S. Sofat, *Malware Analysis and Classification: A Survey*, Journal of Information Security, 2014, pp. 56–64.

[2] A. Shabtai, R. Moskovitch, Y. Elovici, C. Glezer, *Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A state-of-the-art survey*, Information Security Technical Report, 2009.

[3] A. Moser, C. Kruegel, E. Kirda, *Limits of Static Analysis for Malware Detection*, Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC), 2007.

[4] A. Srivastava, A. Lanzi, J. Griffin, D. Balzarotti, *Operating System Interface Obfuscation and the Revealing of Hidden Operations* Holz T., Bos H. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2011. Lecture Notes in Computer Science, vol 6739. Springer, Berlin, Heidelberg.

[5] W.W. Cohen, *Fast Effective Rule Induction*, Machine Learning: Proceedings of the Twelfth International Conference, Lake Taho, California, 1995.

[6] M.G. Schultz, E. Eskin, F. Zadok, S.J. Stolfo, *Data Mining Methods for Detection of New Malicious Executables*, IEEE Symposium on Security and Privacy, 2001.

[7] J.Z. Kolter, M.A. Maloof, *Learning to Detect Malicious Executables in the Wild*, Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '04 - 2004.

[8] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, Y. Elovici, *Unknown Malcode Detection via Text Categorization and the IMbalance Problem*, IEEE International Conference on Intelligence and Security Informatics, 2008.

[9] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman , S. Dolev, Y. Elovici, *Unknown Malcode Detection Using OPCODE Representation*, European Conference on Intelligence and Security Informatics, 2008.

[10] L. Nataraj, S. Karthikeyan, G. Jacob, B.S. Manjunath, *Malware Images: Visualization and Automatic Classification*, Proceedings of the 8th International Symposium on Visualization for Cyber Security, 2011.

[11] D. Kong, G. Yan, *Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification*, Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013.

[12] C. Willems, T. Holz, F. Freiling, *CWSandbox: Towards Automated Dynamic Binary Analysis*, IEEE Security and Privacy, 2007.

[13] K. Rieck, P. Trinius, C. Willems, T. Holz, *Automatic Analysis of Malware Behavior using Machine Learning*, Journal of Computer Security, 2010.

[14] K. Rieck, T. Holz, C. Willems, P. Dussel, P. Laskov, *Learning and Classification of Malware Behavior*, Zamboni D. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2008. Lecture Notes in Computer Science, vol 5137. Springer, Berlin, Heidelberg

[15] M.F. Zolkipli, A. Jantan, *An Approach for Malware Behavior Identification and Classification*, 3rd International Conference on Computer Research and Development, 2011.

[16] M. Bailey, J. Oberheide, J. Andersen, Z. MorleyMao, F. Jahanian, J. Nazario, *Automated Classification and Analysis of Internet Malware*, Lecture Notes in Computer Science Recent Advances in Intrusion Detection, 2007.

[17] T. Lee, J.J. Mody, *Behavioral Classification*, Proceedings of EICAR, 2006.

[18] G. Salton, A. Wong, C.S. Yang, *A Vector Space Model for Automatic Indexing*, Communications of the ACM, 1975.

[19] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, *Distributed Representations of Words and Phrases and Their Compositionality*, Advances in Neural Information Processing Systems, 2013.

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

[21] M.J. Kusner, Y. Sun, N.I. Kolkin, and K.Q. Weinberger, *From Word Embeddings to Document Distances*, Proceedings of ICML, 2015.

[22] R. Rehurek, P. Sojka, *Software Framework for Topic Modelling with Large Corpora*, Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, p45-p50, 2010.