DETECTION OF MALICIOUS VBSCRIPT USING STATIC AND DYNAMIC ANALYSIS WITH RECURRENT DEEP LEARNING

Jack W. Stokes[†] *Rakshit Agrawal*[⋆] *Geoff McDonald*[±]

[†] Microsoft Research, One Microsoft Way, Redmond, WA 98052 USA
[⋆] University of California, Santa Cruz, Santa Cruz, CA 95064 USA
[±] Microsoft Corp., #305 876 14th Ave. W, Vancouver, British Columbia, V5Z 1R1, Canada

ABSTRACT

Attackers have used malicious VBScripts as an important computer infection vector. In this study, we explore a system that employs both static and dynamic analysis to detect malicious VBScripts. For the static analysis, we investigate two deep recurrent models, LaMP (LSTM and Max Pooling) and CPoLS (Convoluted Partitioning of Long Sequences), which process a VBScript as a byte sequence. Lower layers capture the sequential nature of these byte sequences while higher layers classify the resulting embedding as malicious or benign. Our models are trained in an end-to-end fashion allowing discriminative training even for the sequential processing layers. Dynamic analysis allows us to investigate obfuscated VBScripts an additional files which may be dropped during execution. Evaluating these models on a large corpus of 240,504 VBScript files indicates that the best performing LaMP model has a 69.3% true positive rate (TPR) at a false positive rate (FPR) of 1.0%. Similarly, the best CPoLS model has a TPR of 67.9% at an FPR of 1.0%. Our system is general in nature and can be applied to other scripting languages (e.g., JavaScript) as well.

***Index Terms*—** VBScript, Detection, Recurrent Neural Network, Deep Learning

1. INTRODUCTION

Malicious scripts are widely abused by malware authors to infect users' computers. In the current threat landscape, one of the most prevalent types of script malware that Windows users have encountered is VBScript (VBS). VBScript, or Microsoft Visual Basic Scripting Edition, is an active scripting language originally designed for Internet Explorer and the Microsoft Internet Information Service web server [1].

While a wide range of different machine learning models have been proposed for detecting malicious executable files [2], there has been little work in investigating malicious VBScript. Two previous solutions for VBScript are based on static analysis [3, 4]. In addition, deep recurrent models have recently been proposed detecting system API calls in PE files [5, 6, 7], JavaScript [8, 9], and Powershell [10].

There are several challenges posed by trying to detect malicious VBScript. Malicious scripts include obfuscation to hide the malicious content, and often unpack or decrypt the underlying malicious script only upon execution. Complicating this is the fact that the obfuscators, in some cases, are used by both benign and malware files. Thus pure static analysis of the primary script often fails to detect some malicious activity. Another problem is that anti-virus (AV) automation systems such as sandboxing environments are designed primarily to handle Windows Portable Executable (PE) files (*e.g.*, .exe and .dll). Another problem is that anti-virus (AV) analysts typically spend the majority of their time authoring new signatures for executable malware (*e.g.*, .exe, .dll). Accordingly, the number of labeled script files is typically much lower than for executable files.

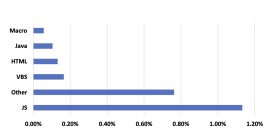
In this paper, we propose VbsNet, a deep recurrent neural classification system which can be trained to detect malicious VBScript using a combination of both static and dynamic analysis. We first use a production anti-virus engine to dynamically execute a script in a sandboxed environment inside of the engine. This allows the AV engine to safely analyze any obfuscated scripts or child scripts which are dropped during script execution without infecting the computer.

We investigate two different models for the task of detecting malicious VBScript using static analysis on the resulting files after dynamic analysis. Both models encode sequential information using one or more long short-term memory (LSTM) layers. The LSTM and Max Pooling (LaMP) model follows a two-stage approach where the first stage learns a language model for the individual characters in the script content. Next, the second stage includes a, potentially deep, neural network for the final classification of the script as malicious or benign. To allow the processing of longer script files, we next investigate the Convoluted Partitioning of Long Sequences (CPoLS) model which adds an additional layer consisting of a one-dimensional convolutional neural network. Since our models operate directly on the script content encoded as bytes, they do not require careful and potentially computationally expensive feature engineering proposed by other solutions.

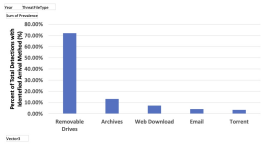
VbsNet is the first deep learning model which has been proposed for detecting malicious VBScript. LaMP and CPoLS have recently been proposed for detecting malicious JavaScript in the ScriptNet model [9]. However, ScriptNet only does static analysis of the file. Since the ScriptNet model does not employ dynamic analysis, it cannot detect malicious child scripts which are dropped during execution. In addition, ScriptNet must deal with obfuscated files. Agrawal, *et al.* [11] also used LaMP and CPoLS for detecting malicious Windows portable executable files. In this work, we show that these models are also effective for detecting malicious VBScript files. The main contributions of this paper include the following. 1) We propose the first deep neural network models for the detection of malicious VBScript. 2) We evaluate these models on a large corpora of VBScript files. 3) We demonstrate that these models can effectively detect malicious VBScript files.

2. MOTIVATION

With advances in browser and operating system security making browser exploit attacks more difficult, miscreants are instead relying on social engineering attacks. Figure 1a indicates the percentage



(a) Percentage of different non-PE file detections in January-September 2019 (remaining 97.65% are PE files).



(b) Arrival methods for malicious VBScript files detected from January through September 2019.

of all, non-PE files detected in the Windows Defender anti-malware product's telemetry for the first nine months of 2019. This figure indicates that VBScript was the second most prevalent type of detected scripts found in the telemetry data. Since the remaining 97.65% of the detections are for PE files, malicious scripts are still a small minority of the detected files in the wild.

Figure 1b illustrates the identified attack methods of VBScript based on telemetry data from January 1, 2019 through September 30, 2019. The main threat vector of malicious VBScript is removable drives followed closely by archives. Web downloads, email and bit torrent play a smaller role in VBScript attacks, but they were still important threat vectors. Through September, Web downloads were the third most prevalent threat vector, so this likely motivated Microsoft to recently disable VBScript execution in Internet Explorer.

3. SYSTEM

Figure 2 presents an overview of VbsNet, our proposed neural VBScript classification system. A labeled collection of malicious and benign VBScripts are first scanned with the Microsoft Windows Defender anti-malware engine. During this scanning operation, the script is emulated and unpacked, and may drop one or more additional scripts. Each child script is also emulated and unpacked which may generate even more scripts. This process continues until all scripts have been extracted and scanned.

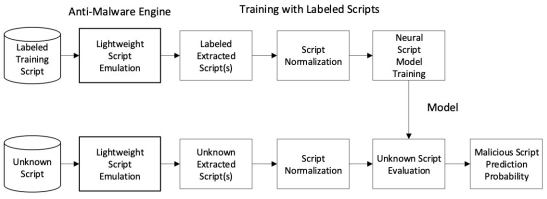


Fig. 2: Overview of the VbsNet neural VBScript classification system.

These scripts are next normalized. All whitespace characters, except line breaks, are first removed. Next the text is standardized to lowercase and converted to the US-ASCII character set. Any characters which are not included in the US-ASCII character set, such as non-English language characters, are replaced by the constant character '?'.

Before training the model, each normalized script is written to the file system. To avoid storing malicious content on the hard drive, the characters are next encoded by their numeric ASCII encoding (*e.g.*, '97' for the character 'a') delimited by commas. This delimited, encoded sequence data is then used to train the neural script malware model.

To evaluate an unknown file, the system uses the trained model to produce a prediction which indicates the probability that the unknown VBScript is malicious.

4. MODELS

Dynamic analysis of VBScript files allows our system to use information hidden in the VBScript's unpacked content to learn its malicious nature. In this section, we discuss our models which can capture and learn the malicious intent of VBScript files using neural classifier models and sequential learning.

Translation to Sequences: The raw scripts can be considered to be documents containing a limited vocabulary set. As such, the VBScripts are long ordered sequences of encoded characters. For normalized VBScript files, we define our vocabulary as the set of all possible bytes (8-bits). This leads to a vocabulary of size 256. Each normalized VBScript, therefore, is a sequence of these bytes.

Model Architectures: In our experiments for sequential learning, we employed two neural model architectures. The primary difference in these two architectures is their resilience against long length sequences.

LSTM and Max Pooling: In the LSTM and Max Pooling (LaMP) architecture, illustrated in Figure 3, we first use an embedding layer, EMBEDDING, to process the input byte sequence B. Since each element in B corresponds to a byte from the vocabulary, it is symbolic in nature. We use the embedding layer to transform each byte into a dense vector (*i.e.*, an embedding) which captures relatedness among different bytes, thereby assisting the overall model in learning. The sequence of embeddings E is then passed through multiple LSTM layers stacked on top of each other. The LSTM generates representations for each element in the input sequence as H_L. In order for us to perform classification on the sequence and identify its hidden malicious content, we transform the sequence H_L into a vector highlighting significant information, while reducing its dimensionality. For this purpose, we use a temporal, max pooling layer, MAXPOOL1D, as proposed by Pascanu *et al.* [7]. Given an input vector sequence $S = [s_0, s_2, \dots s_{M-1}] \in S$ of length M, where each vector $s_i \in \mathbb{R}^k$ is a k-dimensional vector, MAXPOOL1D computes an output vector $s_{MP} \in \mathbb{R}^k$ as $s_{MP}(k) = \max(s_0(k), s_1(k), \cdots s_{M-1}(k))$.

We pass the sequence H_L through MAXPOOL1D to obtain vector h_L. Next, h_L is passed through one or more dense neural layers employing a rectified linear (RELU) nonlinear activation function. This helps learn an additional layer of weights before performing the final prediction. The RELU activation vector is finally used by a sigmoid layer to generate the final probability p_m indicating if the VBScript is malicious or benign. We can formally define LAMP on an input byte sequence B as:

$$\begin{aligned} E &= \text{EMBEDDING}(B) \\ H_L &= \text{LSTM}(E) \\ h_L &= \text{MAXPOOL1D}(H_L) \\ h_{CL} &= \text{RELU}(W_L * h_L) \\ \mathbf{p_m} &= \sigma(W_D * h_{CL}) \end{aligned} \tag{1}$$

where W_L is the weight matrix for the dense RELU hidden layer, and W_D is the weight matrix for the final sigmoid classification layer.

While LaMP provides a simple model to capture sequences directly, it is limited by the length of the input sequences. As the length

of input sequence B increases, the model becomes both difficult to train and more memory-intensive. In the case of detecting malicious content, long sequences can often separate two or more bytes far from each other even when their combined presence is a cause of the malicious intent. When learning directly on a sequence, it is possible for the model to lose the context of an identified byte earlier in the sequence when processing a new byte at a larger distance. To cope with such problems in detection, we therefore, investigate another architecture called Convoluted Partitioning of Long Sequences (CPoLS).

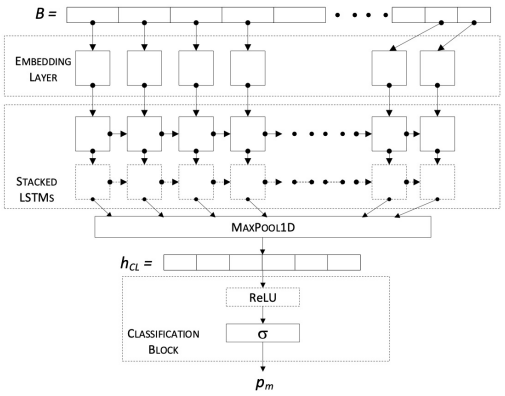


Fig. 3: LaMP model for detecting malicious VBScript files.

Convoluted Partitioning of Long Sequences: Convoluted Partitioning of Long Sequences (CPoLS) is a neural model architecture designed specifically to extract classification information hidden deep within long sequences. In this model illustrated in Figure 4, we process the input sequence in parts by splitting it first into smaller pieces of fixed length. By performing this step, we generate a sequence of multiple partitions, each of which is a sequence in itself of a smaller length.

We use Convolutional Neural Networks (CNNs) [12] in this model, along with the other LaMP modules. CNNs are widely used in computer vision [13, 14], and they have also recently shown success in sequential learning domains as well [15, 16].

Given an input byte sequence B, the model first splits it into a partitioned list C containing several small subsequences $c_i \in C$ where i is the index of each partition in C. To translate the bytes in these sequences from symbols to dense vectors, we pass them through an embedding layer, EMBEDDING, and obtain sequence E, where each element $e_i \in E$ corresponds to the sequence of embeddings for partition c_i in C. Each of these partitions e_i, are now separately processed, while still maintaining their overall sequential nature. We call this method RECURRENTCONVOLUTIONS. In this method, we pass each partition e_i through the one-dimensional CNN, CONV1D, which applies multiple filters on the input sequence and generates tensor e_i^χ representing the convoluted output of vector sequence e_i. χ refers to the sequence with CONV1D performed on it. The combined list of these convolved partitions e_i^χ is referred to as E^χ. In RECURRENTCONVOLUTIONS, we then reduce the dimensionality of e_i^χ by performing a temporal max pooling MAXPOOL1D (not shown). MAXPOOL1D takes a tensor input e_i^χ and extracts a vector e_i' from it. Similarly, we apply RECURRENTCONVOLUTIONS on each partition e_i to obtain the updated vectors e_i'. These vectors e_i' are finally combined in the same order to create an updated sequence E' of learned partition representations. With the help of partitioning, the length of E' is also limited to a trainable length.

Script Model	Parameter	Description	Value
LaMP	$B_{VBS,LaMP}$	Minibatch Size	100
LaMP	$H_{VBS,LaMP}$	LSTM Hidden Layer Size	1500
LaMP	$E_{VBS,LaMP}$	Embedding Layer Size	128
CPoLS	$B_{VBS,CPoLS}$	Minibatch Size	100
CPoLS	$H_{VBS,CPoLS}$	LSTM Hidden Layer Size	1500
CPoLS	$E_{VBS,CPoLS}$	Embedding Layer Size	128
CPoLS	$W_{VBS,CPoLS}$	CNN Window Size	10
CPoLS	$S_{VBS,CPoLS}$	CNN Window Stride	5
CPoLS	$F_{VBS,CPoLS}$	Number of CNN Filters	128

Table 1: Settings for the various model parameters.

At this stage, the model uses sequence E' as an input to the LaMP model and learns the probability p_m that the VBScript is malicious. Therefore, we use a combination of an LSTM, a second MAXPOOL1D layer, dense RELU activations, and a final sigmoid layer for generating the prediction p_m on the new input sequence E'. Formally, we define the CPOLS model as:

$$
\begin{aligned}
C &= \text{PARTITION}(B) \\
E &= [\text{EMBEDDING}(c_i) \quad \forall c_i \in C] \\
E^\chi &= [\text{CONV1D}(e_i) \quad \forall e_i \in E] \\
E' &= [\text{MAXPOOL1D}(e_i^\chi) \quad \forall e_i^\chi \in E^\chi] \\
p_m &= LaMP(E')
\end{aligned}
\tag{2}
$$

Such a model is resilient to extremely long sequence lengths and can also find malicious objects hidden very late in the sequence.

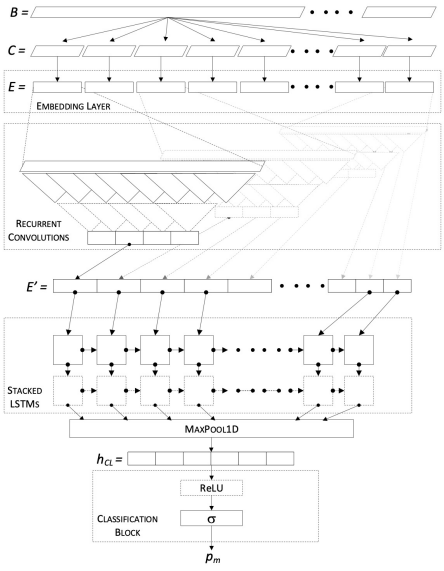


Fig. 4: Convoluted Partitioning of Long Sequences (CPoLS) model for detecting malicious VBScript files.

End-to-End Learning: To train the models described above, we perform an end-to-end learning process. Since the data available to

us is in the form of a sequence and an associated binary label, we need to train the entire model, solely from this label. In end-to-end learning, we pass each sequence B through all layers of our model to derive the probability p_m. Using this probability, with the true label $L \in \{0, 1\}$, we measure the cross-entropy loss $\mathcal{L}$. This loss is used to compute the gradients required for updating the weights in each layer of the model. Therefore, we simultaneously learn all the parameters for the primary classification objective.

5. EXPERIMENTAL RESULTS

We next evaluate the performance of the proposed neural VBScript malware classifier models on files collected from Microsoft's production malware infrastructure. We first start by describing the experimental setup used to generate the results. We then evaluate the LaMP and CPoLS models trained on our large collection of VBScript files.

Datasets: Our anti-virus partners provided the first 1000 bytes of 240,504 VBScript files which contained 66,028 malicious and 174,476 benign scripts. We randomly assigned these scripts into training, validation, and test sets containing 168,353, 24,050, and 48,101 samples, respectively. The labels are obtained from the production antimalware detection system.

Experimental Setup: All the experiments were performed using Keras [17] with the TensorFlow [18] backend. The models were trained and evaluated on a cluster of NVIDIA K40 graphical processing unit (GPU) cards. All models were trained with a maximum of 15 epochs, but early stopping was employed if the model fully converged before reaching the maximum number of epochs. All LaMP models are trained and tested using the first 200 bytes of the VBScript files, while the CPoLS models are evaluated using the first 1000 bytes.

We did hyperparameter tuning of the various input parameters for both types of VBScript models, and the results are summarized in Table 1. With these settings, we evaluate the classification error rate on the test set for the VBScript dataset.

Model Performance: We evaluate the LaMP and CPoLS models for VBScript in Figures 5a and Figure 5b, respectively. The simplest LaMP and CPoLS VBScript models with a single LSTM layer and classifier hidden layer offer the best, or nearly the best, performance compared to the more complex models. At an FPR of 1.0%, the TPR for the LaMP model is 69.3% with $L_{VBS,LaMP} = 1, C_{VBS,LaMP} = 1$. Similarly, CPoLS yields a TPR of 67.1% with $L_{VBS,CPoLS} = 1, C_{VBS,CPoLS} = 1$ at this FPR = 1.0%. Thus, the LaMP models trained with only the first 200 bytes are able to outperform the the CPoLS models which are trained with the first 1000 bytes.

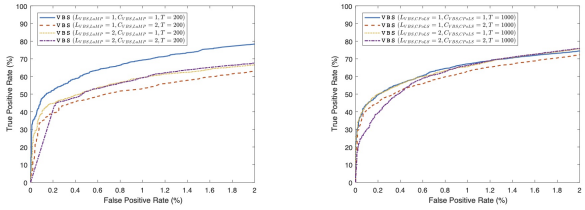
6. RELATED WORK

VBScript: The detection of malicious VBScript has been an understudied problem, but there have been a few works which consider this script type. Kim et al. [3] take a static analysis, graph-based approach and search for conceptual graphs which are similar to those containing malicious VBScript files. Wael *et al.* [4] propose a number of different classifiers to detect malicious VBScript including Logistic Regression, a Support Vector Machine with an RBF kernel, a Random Forest, a Multilayer Perceptron, and a Decision Table. In [19], Zhao and Chen detect malicious applets, JavaScript and VBScript based on a method which models immunoglobulin secretion.

JavaScript: Our work is most closely related to ScriptNet [9] by Stokes, *et al.* which also uses LaMP and CPoLS for the detection of JavaScript. In addition to using the model on a different type of script, ScriptNet evaluates the script using only static analysis. VbsNet uses a combination of both static and dynamic analysis. Deep learning models, sparse random projections, logistic regression and auto-encoders were used to detect malicious Javascript by Wang *et al.* [8]. A statistical n-gram language model was proposed for the purpose of detecting malicious JavaScript in [20]. Other papers which investigate the detection of malicious JavaScript include [21, 22, 23, 24, 25].

Other File Types: A number of deep learning models have been proposed for detecting malicious PE files including [5, 26, 27, 6, 7]. Agrawal *et al.* [11] also use LaMP and CPoLS for the detection of malicious PE files using system API calls. A character-level CNN has been proposed for detecting malicious PE files [5] and Powershell script files [10].

Architecture: An approach for learning from images presented in RCNN [28] uses an architecture similar to CPoLS but performs 2-dimensional convolutions on images, and derives their vector representation via dense layers, before passing it through an RNN. We, instead use 1-dimensional convolutions feeding directly into the RNN, followed by max-pooled learning over the sequence outputs.



(a) LaMP (b) CPoLS

Fig. 5: ROC curves for various VBScript models.

7. CONCLUSIONS

Our analysis shows that malicious VBScript was the second most prevalent type of malicious script encountered by Windows users for the first nine months of 2019. In this work, we investigate combining static analysis and dynamic analysis to help detect malicious VBScript. Dynamic analysis allows us to detect additional files which are dropped during execution of obfuscated commands. The results show that the LaMP model, which employs a full LSTM for the sequence model, is able to outperform the CPoLS architecture for the task of VBScript detection using only 20% of the initial bytes in the file. However, CPoLS scales much better, and we expect CPoLS to perform better with more data since the number of bytes that the LaMP model can process is limited due to its high computational and memory resource requirements. Analyzing the model's performance on VBScript provides important validation for the model's applicability for malicious scripts in general. Even though VBScript has recently been disabled in Internet Explorer, our proposed system is general in nature and can be applied to other scripting languages such as JavaScript. Results of the proposed system on JavaScript can be found in our extended technical report [29].

8. REFERENCES

[1] Microsoft, "Vbscript," https://msdn.microsoft.com/en-us/ library/t0aew7h6.aspx.

[2] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," pp. 55–64, 2014.

[3] Sungsuk Kim, Chang Choi, Junho Choi, Pankoo Kim, and Hanil Kim, "A method for efficient malicious code detection based on conceptual similarity," in *International Conference on Computational Science and Its Applications (ICCSA)*, 2006, vol. 3983, pp. 567–576.

[4] D. Wael, A. Shosha, and S. G. Sayed, "Malicious vbscript detection algorithm based on data-mining techniques," in *2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems 2017 Intl Conf on New Paradigms in Electronics Information Technology (PEIT)*, Nov 2017, pp. 112–116.

[5] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 2482–2486.

[6] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer International Publishing, 2016, pp. 137–149.

[7] 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)*, April 2015, pp. 1916–1920.

[8] Yao Wang, Wan dong Cai, and Peng cheng Wei, "A deep learning approach for detecting malicious javascript code," *Proceedings of Security and Communication Networks*, vol. 11, no. 9, pp. 1520–1534, 2016.

[9] Jack W. Stokes, Rakshit Agrawal, Geoff McDonald, and Matthew Hausknecht, "Scriptnet: Neural static analysis for malicious javascript detection," in *Proceedings of the Military Communications Conference (MILCOM)*, 2019.

[10] D. Hendler, S. Kels, and A. Rubin, "Detecting Malicious PowerShell Commands using Deep Neural Networks," *ArXiv e-prints*, Apr. 2018.

[11] Robust Neural Malware Detection Models for Emulation Sequence Learning, "Rakshit agrawal and jack w. stokes and mady marinescu and karthik selvaraj," in *Proceedings of the Military Communications Conference (MILCOM)*, 2018.

[12] Yann LeCun and Yoshua Bengio, "Convolutional networks for images speech and time series," 1995.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[15] Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin, "A convolutional encoder model for neural machine translation," *CoRR*, vol. abs/1611.02344, 2016.

[16] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin, "Convolutional sequence to sequence learning," *CoRR*, vol. abs/1705.03122, 2017.

[17] François Chollet et al., "Keras," `https://github.com/fchollet/keras`, 2015.

[18] Martín Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[19] H. Zhao and W. Chen, "A web page malicious script detection method inspired by the process of immunoglobulin secretion," in *2010 International Symposium on Intelligence Information Processing and Trusted Computing*, Oct 2010, pp. 241–245.

[20] Anumeha Shah, "Malicious JavaScript Detection using Statistical Language Model," *Master's Projects*, p. 70, 2016.

[21] Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto, "Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references," in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, New York, NY, USA, 2014, AISec '14, pp. 47–57, ACM.

[22] D. Liu, H. Wang, and A. Stavrou, "Detecting malicious javascript in pdf through document instrumentation," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 100–111.

[23] Kristof Schütt, Marius Kloft, Alexander Bikadorov, and Konrad Rieck, "Early detection of malicious behavior in javascript code," in *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, New York, NY, USA, 2012, AISec '12, pp. 15–24, ACM.

[24] Wei-Hong Wang, Yin-Jun Lv, Hui-Bing Chen, and Zhao-Lin Fang, "A static malicious javascript detection using svm," in *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering*, 2013.

[25] Wei Xu, Fangfang Zhang, and Sencun Zhu, "Jstill: Mostly static detection of obfuscated malicious javascript code," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, New York, NY, USA, 2013, CODASPY '13, pp. 117–128, ACM.

[26] George E. Dahl, Jack W. Stokes, Li Deng, and Dong Yu, "Large-scale malware classification using random projections and neural networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[27] Wenyi Huang and Jack W. Stokes, "Mtnet: A multi-task neural network for dynamic malware classfication," in *Proceedings of Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2016, pp. 399–418.

[28] Rodrigo Carrasco-Davis, Guillermo Cabrera-Vives, Francisco Förster, Pablo A. Estévez, Pablo Huijse, Pavlos Protopapas, Ignacio Reyes, Jorge Martínez-Palomera, and Cristóbal Donoso, "Deep learning for image sequence classification of astronomical events," *Publications of the Astronomical Society of the Pacific*, vol. 131, no. 1004, pp. 108006, Sep 2019.

[29] Jack W. Stokes, Rakshit Agrawal, and Geoff McDonald, "Neural classification of malicious scripts: A study with javascript and vbscript," *CoRR*, 2018.