An Improved Tool for Detection of XSS Attacks by Combining CNN with LSTM*

Caio Lente, Roberto Hirata Jr., Daniel Macêdo Batista

[1]Department of Computer Science
University of São Paulo
São Paulo, Brazil
{lente, hirata, batista}@ime.usp.br

Abstract. *Cross-Site Scripting (XSS) is still a significant threat to web applications. By combining Convolutional Neural Networks (CNN) with Long Short-Term Memory (LSTM) techniques, researchers have developed a deep learning system called 3C-LSTM that achieves upwards of 99.4% accuracy when predicting whether a new URL corresponds to a benign locator or an XSS attack. This paper improves on 3C-LSTM by proposing different network architectures and validation strategies and identifying the optimal structure for a more efficient, yet similarly accurate, version of 3C-LSTM. The authors identify larger batch sizes, smaller inputs, and cross-validation removal as modifications to achieve a speedup of around 3.9 times in the training step.*

1. Introduction

Cross-Site Scripting (XSS) vulnerabilities were first identified around 1996 in the early days of the Internet [2]. With the introduction of JavaScript to web applications, bad actors noticed that it was possible to embed HTML frames on a web page and then read information from one frame into the other with the execution of arbitrary code. This allows the stealing of passwords, cookies, and additional sensitive information. Even after Netscape implemented the "same-origin policy" — which prevented JavaScript from one website from reading data from another — attackers continued to develop strategies to disregard this restriction.

Now that web applications are all but pervasive, XSS attacks continue to be a considerable security risk. In 2017 [5], XSS vulnerabilities were found in around two-thirds of all applications. They are continually being discovered, and there are even freely available automated tools for exploiting them. A successful attack might result in serious security violations for the host of the website and the user accessing it. More modern exploits can inject arbitrary code into user input fields and manipulate the page, take control of the user's account, and even cause a denial of service [1].

XSS is usually divided into three distinct categories: reflected, stored, and DOM-based [5]. The first involves the victim's browser executing unvalidated and unescaped input. The second happens when the malicious input is stored by the attacked application

*Source code: `https://github.com/clente/3clstm/` Documentation: `https://github.com/clente/3clstm/blob/master/README.md` Video: `https://youtu.be/RU-1kTjiFNU` — These and all the other URLs referenced in this paper were last accessed on July 19th, 2021.

and then executed by visiting users (who pass on their privileges to the attacker). The third occurs when the application dynamically includes attacker-controllable data.

Given the prevalence and potential risks involved in XSS attacks, developing techniques and strategies to stop them is a worthwhile effort. Just as XSS attacks, XSS detection can also be divided into three categories: static, dynamic, and hybrid. Static methods analyze web applications' code in search of vulnerabilities. Dynamic methods inject code into the website to observe whether there is a vulnerability, and hybrid methods combine the other two techniques. In general, dynamic analysis methods have a high false-negative rate because they can never cover all cases. One way to circumvent the problem is to take advantage of the recent advances in machine learning technology and develop a "smarter" detector that can extract latent characteristics of XSS attacks. One such new method is 3C-LSTM proposed in [3].

The architecture developed by 3C-LSTM's creators is innovative because it combines in the same algorithm a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM). In its preliminary tests, the neural network reached a precision rate of 99.88% and a recall rate of 99.04%, which is groundbreaking for a dynamic detection algorithm.

This paper aims to improve 3C-LSTM even further to showcase the new possibilities this technology brings to the table and establishes it as a viable XSS detection method. We accomplish this by proposing an improved version of 3C-LSTM, obtained by tweaking the hidden layers of the deep network, updating the algorithm to take advantage of TensorFlow 2's new functionality, and modifying the preprocessing step of 3C-LSTM to enhance word2vec's embedding. Our improvements led to a maximum speedup of around 3.9 times in the training step. In this best case in terms of speedup, the accuracy was only 0.11% lower than that from the original proposal, used as a baseline. The improvement in the training step is extremely important, since it can reduce the needed time to detect an intrusion. The source code for the original 3C-LSTM and the improvements presented in this paper are all publicly available[1]. Besides the algorithm itself, the repository also contains all the data used to train the model, and the results of the experiments described in Section IV.

This paper is organized into five other sections. Section II discusses related works such as different XSS detection algorithms, and Section III focuses on 3C-LSTM and how to improve it. Sections IV and V describe the experiments that were conducted and their results, respectively. Lastly, Section VI presents the conclusions of the paper and possible further improvements.

2. Related Work

The basis for 3C-LSTM is presented in DeepXSS [6], a dynamic XSS detection technology based on deep learning and developed in 2018 by one of 3C-LSTM's authors. It achieved a precision rate of 99.5% and a recall rate of 97.9% in a real dataset, demonstrating this new technique's potential.

Other machine learning-based methods [7] [8] [9] [10] have been developed in recent years, but before DeepXSS, all of them depended on explicitly designed features.

[1]GitHub repository: `https://github.com/clente/3clstm`

This means that these methods are prone to the same problems as typical dynamic analysis methods, namely a high false-negative rate caused by the inability to extract deeper level features that can better represent the data.

As for non-machine learning methods, they suffer from the same shortcomings as their modern counterparts and more: high false-negative rates, high time overhead (as a result of the increase in the number of XSS payloads), and sometimes the impossibility of using them against all types of XSS attacks [4].

To the best of our knowledge, 3C-LSTM, the successor of DeepXSS, is one of the best dynamic XSS detection methods. Unlike its predecessor, 3C-LSTM is trained using an open dataset, which allows it to be improved by other researchers. This paper improves 3C-LSTM by identifying the optimal structure for a more efficient, yet similarly accurate, version of the algorithm.

3. Architecture and Main Functionalities of the Improved Version of 3C-LSTM

3C-LSTM [3] works much in the same way as DeepXSS [6], with very few modifications. It consists of four main steps: processing input data, Word2vec transform, convolution, and LSTM. The processing step prepares the data for ingestion by Word2vec, which encodes the URLs as vectors. Finally, convolution and LSTM encompass the deep learning step that will classify URLs into XSS attacks or non-XSS. An illustration of 3C-LSTM is presented in Figure 1.

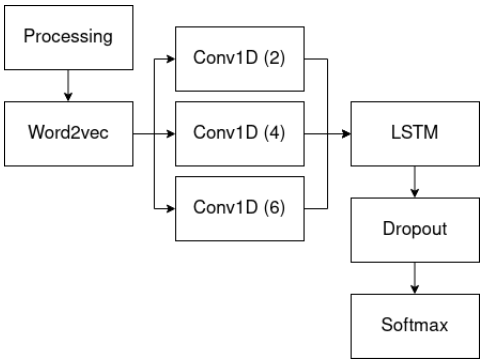

Figure 1. Basic structure of 3C-LSTM

Since URLs have several structures in common with each other, many dissimilarities carry no information at all. For this reason, the processing step aims to smooth out these interferences: all numbers are substituted with “0”, and every locator is replaced with “http://u”. Lastly, the URLs are tokenized using regular expressions that detect structures such as opening tags, closing tags, locators, etc.

The next step in the pipeline is feeding the pre-processed input data to an embedding algorithm. Google introduced Word2vec in 2013 [11] [12] and it has become ubiquitous in text processing applications. This model consists of a neural network that can take a corpus of text and produce a vector space where each word is assigned a unique vector. The main advantage of using Word2vec is that its representations are efficient and preserve context through proximity. There are two different architectures for Word2vec—CBOW (continuous bag-of-words) and continuous skip-gram—, but only CBOW is used

in 3C-LSTM, mainly because it has a much faster training time. Figure 2 illustrates the basic structure of a Word2vec's CBOW: the vector associated with the current word or token is derived from the window around it.

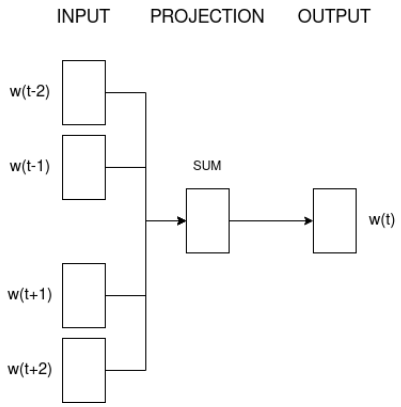

Figure 2. Word2vec's CBOW, based on a figure from [11]

After generating Word2vec's embedding, the next step is feeding these inputs to the actual model. The first part of 3C-LSTM proper is a CNN (convolutional neural network), a type of model introduced around 1990 [13] that implements a multilayer perceptron with regularization. This step has three distinct pathways: all are 1D convolutions, but the first has a window of length 2, the second, of length 4, and the third, of length 6. This means that the convolution kernel, to be convolved with the input, includes more (or less) tokens on each pathway.

Every pathway also includes a step that pads the outputs with zeroes so that all of them have the same final length and a ReLu activation layer. The ReLU (short for Rectified Linear Unit) layer [15] uses a rectifier function, defined as the positive part of its argument, to enable learning.

The outputs of the three pathways are then concatenated and fed to an LSTM model. LSTM stands for Long Short-Term Memory, a recurrent neural network architecture introduced in 1997 [14] that can process an entire sequence of data such as video, audio, or text. Since each LSTM "cell" (see Figure 3) has the capability of processing data sequentially while at the same time retaining its hidden state through time, this architecture is very good at learning from data with unpredictable lags between important events.

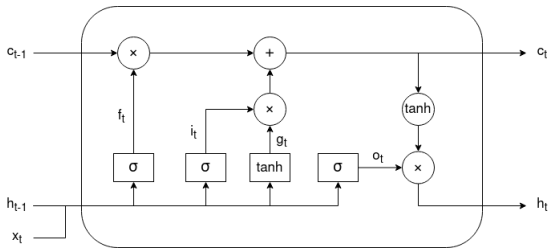

Figure 3. LSTM's cell, based on figure from [6]

After the LSTM step, the last two layers of the model are the Dropout and the Softmax layers. In short, the Dropout layer ignores cells at random during the training

phase of the model to reduce overfitting, while the Softmax layer applies a normalized exponential function to the output of the model to return a probability distribution over predicted output classes from non-normalized vector components.

Once applied to new input, the output of 3C-LSTM is simply the prediction of whether that input corresponds to an XSS attack or not.

3.1. Designing a Better 3C-LSTM

Many changes could be made to 3C-LSTM to improve it, the most obvious ones being porting the code to TensorFlow 2, since it was developed on TensorFlow 1, and restructuring the training step of the model. Launched in late 2019, TensorFlow 2 is the second major release of Google's widely used TensorFlow machine learning framework and includes many improvements over its predecessor, including eager evaluation, better code optimization, and simplified syntax. By porting the code, we also ensure that 3C-LSTM can remain relevant and up to date over TensorFlow's next release cycle.

As for the training step, 3C-LSTM's public code might use a few tweaks. 3C-LSTM currently trains the model in ten steps (taking about 33 minutes each in commodity hardware) with different training dataset partitions. Besides this long routine, the original authors defined each round of training to last only one epoch. By using more epochs and fewer batches, 3C-LSTM's training could take much less time with no impact in accuracy.

Another improvement we will consider is changing the pathway configuration for the convolution step, testing both more and fewer pathways to see whether this impacts on performance. 3C-LSTM will also be refactored, and minor spelling mistakes will be corrected, but this has no relation to the model's accuracy or training time.

All the changes detailed in this subsection are implemented in the improved version of 3C-LSTM presented in this paper. The overall training methodology, however, remains the same: training data is fed into the algorithm, which tunes its internal parameters until it converges on a combination whose output satisfactorily approximates the reference values. Naturally, changing this would mean changing the overarching structure of the algorithm.

4. Experimental Design

The dataset used for testing the new versions of 3C-LSTM is the same one made available by its authors [3]. It consists of a sample of 33,426 XSS attacks extracted from the XSSed database (http://www.xssed.com/) and 31,407 regular URLs extracted from the DMOZ database (http://dmoztools.net/). XSSed is a website that collects data from known XSS attacks and vulnerabilities, compiling the URLs into one easy to access archive. DMOZ is, according to is authors, "the largest, most comprehensive human-edited directory of the Web" and contains a curated hierarchical scheme of links to websites. With these two labeled datasets, 3C-LSTM can be trained to predict whether a new URL represents a benign locator or an XSS attack.

All tests described in this paper were conducted on a 12-core Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz with 64GB RAM. For reproducibility purposes, instead of a native Python-TensorFlow installation, we opted to run the tests inside a virtual machine with stable releases of these software: Python 3.7.6 and TensorFlow 2.1.0. The

virtual machine template was optimized for machine learning workloads and fully configured out of the box, which means externalities were all but eliminated from the testing environment.

Each performance/accuracy experiment consists of one 3C-LSTM run, from training to testing, where both training time and prediction accuracy are collected. After four rounds of experiments, the results were compiled and averaged. Multiple distinct configurations were tried to find the best path for improving 3C-LSTM: Batch sizes (50, 200, 500); Number of epochs (2); Train/test split (2/1, 1/1); Row sampling (25%, 50%); and CNN pathways (none, only 1, only 2, only 3).

By default, 3C-LSTM uses batches of size 100, 1 epoch, and no row sampling. Rather than splitting the dataset into a training partition and a testing partition, it uses 10-fold cross-validation (which is why the original algorithm trains the model in ten steps). For fairness, in the tests where 10-fold is used instead of a train/test split, only the first fold's training time is counted; this is possible because the other nine steps did not affect the accuracy but lengthened training time considerably. Note that all tests, except for the control (the original 3C-LSTM), were run after porting the code to TensorFlow 2.

5. Results

The training time and the accuracy of 3C-LSTM in the different configurations explored in this paper are presented on Table 1. The metrics are presented in terms of the average and the standard deviation. The values of the first line of the table, labeled as **Control**, were obtained by applying 3C-LSTM without the modifications proposed by us, serving as a baseline. The best result in terms of training time and the best result in terms of accuracy are highlighted in gray.

As it is possible to see in Table 1, 3C-LSTM has demonstrated itself as a very robust algorithm, achieving roughly the same prediction accuracy independently of the configuration (including **No Pathway**, where the convolution step was removed). In the best case configuration (**Row Sampling 50%**), the accuracy was 99.36%. In the worst case configuration (**Row Sampling 25%**), the accuracy was 99.17%, a reduction of only 0.19%. From the training time data, it is clear that larger batch sizes (**Batch 200** and **Batch 500**) were more efficient than smaller ones (**Batch 50** and **Control**). On the other hand, the number of epochs (**2 Epochs**) and the CNN pathway architecture (**Pathway 1**, **Pathway 2**, and **Pathway 3**) barely affected this metric, suggesting that maintaining only one epoch and leaving only the LSTM part of the network might be a possibility.

The most interesting results, however, were the ones involving row sampling and train/test split. Row sampling deliberately reduces the size of the input dataset, allowing for much faster training time with no apparent impact on accuracy, but it ignores from 50% to 75% of the available data; the considerably faster training time suggests that the model could be retrained at much shorter intervals at the cost of some information loss. The reduction in the training time can be noted by comparing the original 3C-LSTM (**Control** line in Table 1) with the **Row Sampling 25%** configuration. A reduction of 3.9 times was observed. In this best configuration in terms of training time, the accuracy was only 0.11% lower than that from the original proposal. Splitting the data into a training set and a testing set (as opposed to using 10-fold cross-validation) also reduces the training time (Lines **Train/test split 1/1** and **Train/test split 2/1** in Table 1), but discards no data and

renders the 10 step training regimen unnecessary, which makes this an excellent strategy for improving 3C-LSTM.

Table 1. Average Results of Experiments (SD standing for Standard Deviation, and the best results highlighted in gray)

Configuration	Time Avg. (SD) in minutes	Accuracy Avg. (SD) in %
Control	33.77 (0.16)	99.28 (0.06)
Batch 50	35.55 (0.08)	99.34 (0.07)
Batch 200	33.60 (0.10)	99.30 (0.06)
Batch 500	33.34 (0.05)	99.31 (0.11)
2 Epochs	33.71 (0.07)	99.29 (0.05)
Train/test split 1/1	21.30 (4.87)	99.30 (0.07)
Train/test split 2/1	28.82 (7.21)	99.33 (0.05)
Row Sampling 50%	16.93 (0.02)	99.36 (0.10)
Row Sampling 25%	8.54 (0.01)	99.17 (0.25)
No Pathway	29.70 (0.64)	99.22 (0.11)
Pathway 1	30.80 (0.96)	99.20 (0.10)
Pathway 2	31.14 (0.92)	99.31 (0.07)
Pathway 3	31.13 (0.57)	99.25 (0.08)

5.1. Demonstrations

The improved version of 3C-LSTM can be easily demonstrated by cloning the repository available at `https://github.com/clente/3clstm/`. All the experiments reported in this section can be reproduced by following the instructions presented in the video available at `https://youtu.be/RU-1kTjiFNU`.

6. Conclusions

3C-LSTM's accuracy is robust to architectural modifications. Changing the size of its batches, the number of epochs, convolution structure, validation strategy, and even the training dataset's size had basically no effect on accuracy, which always remained above 99%. However, improvements can be achieved in terms of training time. We suggested using larger sized batches, no cross-validation, and sampling the input data whenever possible to improve the algorithm. According to the experiments, these modifications can improve up to 3.9 times in regards to training time, with no statistically significant impact on accuracy. In the future, we plan on testing even more architectures for 3C-LSTM. Tweaking Word2vec's and the model's hyperparameters could marginally improve accuracy, but the most significant contribution would be collecting more up-to-date XSS data to make sure the algorithm handles them with similar effectiveness. We also have plans to test the interactions of the different tested configurations, such as using no cross-validation and also increasing the batch size to 1000.

7. Acknowledgments

This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9. It is also part of the FAPESP proc. 18/22979-2 and proc. 18/23098-0.

References

[1] Hydara, I., Sultan, A. B. M., Zulzalil, H., & Admodisastro, N. (2015). Current state of research on cross-site scripting (XSS)–A systematic literature review. Information and Software Technology, 58, 170-186.

[2] Grossman, J., Fogie, S., Hansen, R., Rager, A., & Petkov, P. D. (2007). XSS attacks: cross site scripting exploits and defense. Syngress.

[3] Boyu Zhang, "Detecting XSS attacks by combining CNN with LSTM", IEEE Dataport, 2019. [Online]. http://dx.doi.org/10.21227/css6-ds36. Accessed: Apr. 23, 2020.

[4] Liu, M., Zhang, B., Chen, W., & Zhang, X. (2019). A Survey of Exploitation and Detection Methods of XSS Vulnerabilities. IEEE Access, 7, 182004-182016.

[5] Wichers, D., & Williams, J., "Owasp top-10 2017", OWASP, 2017.

[6] Fang, Y., Li, Y., Liu, L., & Huang, C. (2018, March). DeepXSS: Cross site scripting detection based on deep learning. 2018 International Conference on Computing and Artificial Intelligence (pp. 47-51).

[7] Gupta, S., & Gupta, B. B. (2017). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. International Journal of System Assurance Engineering and Management, 8(1), 512-530.

[8] Goswami, S., Hoque, N., Bhattacharyya, D. K., & Kalita, J. (2017). An Unsupervised Method for Detection of XSS Attack. IJ Network Security, 19(5), 761-775.

[9] Vishnu, B. A., & Jevitha, K. P. (2014, October). Prediction of cross-site scripting attack using machine learning algorithms. In Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing (pp. 1-5).

[10] Rathore, S., Sharma, P. K., & Park, J. H. (2017). XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs. JIPS, 13(4), 1014-1028.

[11] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[12] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

[13] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Advances in neural information processing systems (pp. 396-404).

[14] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[15] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature, 405(6789), 947-951.

[16] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.