Detecting Attacks on Web Applications using Autoencoder

Conference Paper · December 2018

DOI: 10.1145/3287921.3287946

3 authors:

Hieu Mac
Hanoi University of Science and Technology
8 PUBLICATIONS **316** CITATIONS
SEE PROFILE

Duc Quang Tran
Hanoi University of Science and Technology
17 PUBLICATIONS **341** CITATIONS
SEE PROFILE

Hai-Anh Tran
Hanoi University of Science and Technology
55 PUBLICATIONS **583** CITATIONS
SEE PROFILE

Detecting Attacks on Web Applications using Autoencoder

Hieu Mac
Bach Khoa Cybersecurity Centre
HUST
Vietnam
hieumd@soict.hust.edu.vn

Dung Truong
Bach Khoa Cybersecurity Centre
HUST
Vietnam
dung.tt150722@sis.hust.edu.vn

Lam Nguyen
Bach Khoa Cybersecurity Centre
HUST
Vietnam
lam.nt152131@sis.hust.edu.vn

Hoa Nguyen
Department of Information Systems
UET-VNU
Vietnam
hoa.nguyen@vnu.edu.vn

Hai Anh Tran
Bach Khoa Cybersecurity Centre
HUST
Vietnam
anhth@soict.hust.edu.vn

Duc Tran
Bach Khoa Cybersecurity Centre
HUST
Vietnam
ductq@soict.hust.edu.vn

ABSTRACT

Web attacks have become a real threat to the Internet. This paper proposes the use of autoencoder to detect malicious pattern in the HTTP/HTTPS requests. The autoencoder is able to operate on the raw data and thus, does not require the hand-crafted features to be extracted. We evaluate the original autoencoder and its variants and end up with the Regularized Deep Autoencoder, which can achieve an F1-score of 0.9463 on the CSIC 2010 dataset. It also produces a better performance with respect to OWASP Core Rule Set and other one-class methods, reported in the literature. The Regularized Deep Autoencoder is then combined with Modsecurity in order to protect a website in real time. This algorithm proves to be comparable to the original Modsecurity in terms of computation time and is ready to be deployed in practice.

CCS CONCEPTS

• **Security and privacy → Firewalls**;

KEYWORDS

Anomaly Detection, Autoencoder, Web attack detection, Web Application Firewall

ACM Reference Format:
Hieu Mac, Dung Truong, Lam Nguyen, Hoa Nguyen, Hai Anh Tran, and Duc Tran. 2018. Detecting Attacks on Web Applications using Autoencoder. In *The Ninth International Symposium on Information and Communication Technology (SoICT 2018), December 6–7, 2018, Danang City, Viet Nam.* ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3287921.3287946

1 INTRODUCTION

Web attacks are attacks that target the HTTP/HTTPS protocol. The aim is to evade the Web Application Firewall (WAF) and gain an unauthorized access to the proprietary data. There exist hundreds of web attacks, which can be found in practice [11] . These include but are not limited to Directory Traversal, SQL Injection, Broken Authentication and Session Management and Cross-Site Scripting (XSS). Web attacks are a major threat to the Internet with an average of 229,000 incidents, detected every single day in 2016 [3]. Given the prominence of this threat, the cybersecurity community has addressed the attacks on web applications in many different ways. Signature-based approach operates by searching for a known identity to detect malicious patterns. It is not efficient in handling new unknown attacks, whereas maintaining the signature database can be problematic because the attacks have become increasingly sophisticated.

Anomaly-based approach can be treated as a pattern classification problem, whose objective is to distinguish between the abnormal and benign patterns. Ingham et al. [6] described the Deterministic Finite Automata (DFA) induction, which is based on the rules for reducing variability among queries and heuristic for filtering anomalies. Kruegel et al. [8] detected web attacks using the attribute length, character distribution, access frequency, inter-request time delay and structure of the parameters. Nguyen et al. [10] considered 30 features that are relevant for the detection process. The authors also applied a so-called Generic-Feature-Selection to extract the most important features in order to reduce the computational complexity and avoid the problem of curse of dimensionality. Torrano et al. [16] exploited the expert knowledge to determine actions that deviate from the intrusions. The authors worked at the token- rather than the request level by using the length and the structure of a given token. Kozik et al. [7] developed a pattern extraction method for HTTP traffic anomaly detection, which is based on combination of text segmentation and statistical analysis to capture the structure of consecutive requests. Rieck and Laskov [14] highlighted the importance of higher-order *n*-grams in diminishing unknown attacks. Using the *n*-grams and expert knowledge to derive the features has a potential drawback since they can be circumvented by the attackers. It is also rather challenging to derive a new set of features.

Deep learning has recently received an increased interest in computer security. Liang et al. [9] introduced the Recurrent Neural Network (RNN) with LSTM and GRU cell structures to learn the

SoICT 2018, December 6–7, 2018, Danang City, Viet Nam
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6539-0/18/12…$15.00
https://doi.org/10.1145/3287921.3287946

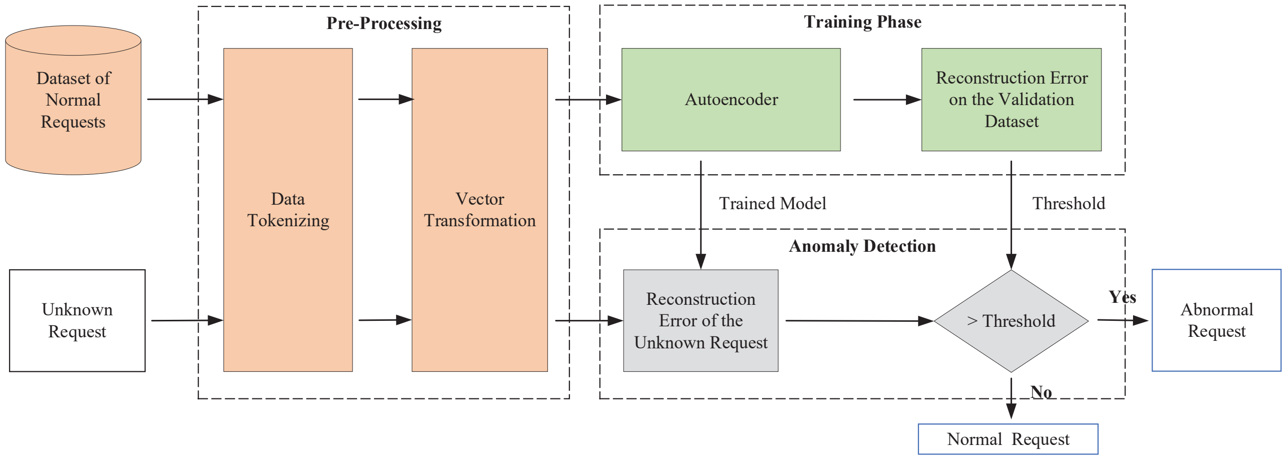

Figure 1: The architecture of the web anomaly detection using Autoencoder.

normal request patterns. The Multilayer Perceptron (MLP) is subsequently applied to detect anomalies based on the outputs of the RNN. Yuan et al. [20] developed a deep learning enabled subspace spectral ensemble clustering approach to divide the anomalies into several specific clusters, such as SQL injection. Wang et al. [18] explored CNN and LSTM to detect web attacks. Although deep learning is observed to produce a better overall accuracy as compared to other traditional methods, it cannot be done with scarce labeled data. Most deep learning architectures are discriminative. They require amount of labor to judge whether a request is legitimate or not, which may not be possible to achieve in practice.

This paper is the continuation of the on-going research on web attacks. Specifically, we present a novel unsupervised learning algorithm, known as autoencoder, whose classification model can be trained using samples exclusively from the normal data class. As is the case with deep learning, the autoencoder is able to operate directly on the raw HTTP/HTTPS requests and does not need the hand-crafted features to be extracted. We evaluate several autoencoder variants on the CSIC 2010 dataset [4] and end up with the Regularized Deep Autoencoder (RDA) [1]. This algorithm demonstrates to achieve a better performance with respect to one-class SVM and the stacked autoencoder when detecting malicious patterns in HTTP/HTTPS queries. We then develop an architecture, where the RDA is combined with the well-known Modsecurity [15] to protect a website in real time. This architecture is proved to be comparable to the original Modsecurity in terms of computation time, while improving the F1-score by up to 13%.

The remainder of this paper is organized as follows. Section 2 details the RDA, its variants and the overall WAF, in which such algorithm is deployed. The empirical evaluation is presented in Section 3. Section 4 is dedicated to conclusion and future works.

2 PROPOSED METHODOLOGY

This section presents the web anomaly detection using autoencoder. The idea is to train the classification model on the requests exclusively from the legitimate data class. We compute the reconstruction error for a given request and render the decision based on a predefined threshold. The architecture regarding the web anomaly detection using autoencoder is shown in Figure 1.

2.1 Data preprocessing

The data preprocessing is based on the URL Tokenizing to collect the method, absolute path and query parameters from the HTTP/HTTPS requests. The aim is to reduce the variability and capture the important information to detect malicious patterns. Liang et al. [9] argued that the anomalies can be found using path and parameter structure of the HTTP GET. We extend their method, not only to preprocess the GET, but the POST and PUT queries. This procedure includes:

- Map the upper-case letters into the corresponding lower-case ones
- Replace numerical values by <NV>, while other string values are substituted by the <SV> token in the query values
- Use <BD> to divide the header and body in the PUT and POST requests
- Put the method at the beginning of the input query

We note that special characters are not changed during the URL Tokenizing. These characters are essential to differentiate the malicious from legitimate sequences. Unlike the work of Liang et al., this paper does not consider the word embedding. Instead, the characters are directly replaced by numbers in accordance with the ASCII coding table. This vector is subsequently normalized using the max-min data scaling and is fed into the autoencoder as described in the following section.

2.2 Autoencoder

Autoencoder [1] is a neural network that is trained to learn a representation for a dataset. It is composed of two different components: an encoder that maps the original input $\mathbf{x}$ to a hidden layer H with a function $\mathbf{h} = f(\mathbf{x})$ and a decoder, whose primary aim is to produce the reconstruction $\mathbf{x'} = g(\mathbf{h})$ [5]. In the simplest form, the autoencoder can be summarized as

$$\mathbf{h} = \sigma(W_1\mathbf{x} + b_1) \tag{1}$$

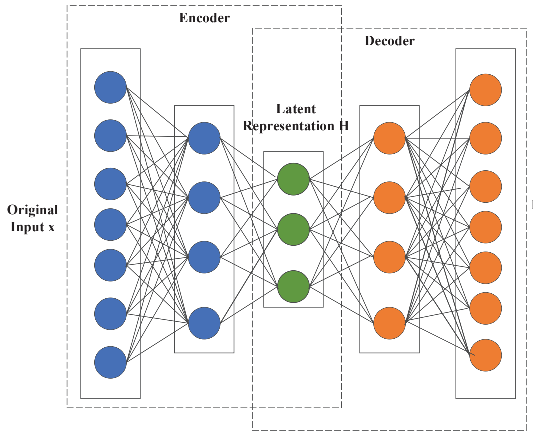

Figure 2: The general autoencoder architecture.

$$\mathbf{x'} = \sigma(W_2\mathbf{h} + b_2) \tag{2}$$

where σ is an activation function and b_i is bias vector. $W_i \epsilon \Re^{I_{di} \times O_{di}}$ is the parameter matrix of the i-th layer, projecting a I_{di} dimensional input into an O_{di} dimensional output. The autoencoder tries to minimize the reconstruction error between the input value **x** and reconstructed value **x'**. The reconstruction error can be computed using either the L2 norm (Eq. 3) or the cross entropy (Eq. 4) as

$$\min \mathcal{L} = \min E(\mathbf{x}, \mathbf{x'}) = \min \|\mathbf{x} - \mathbf{x'}\| \tag{3}$$

$$\mathcal{L}(\mathbf{x} - \mathbf{x'}) = -\sum_{c=1}^{M} x'_c \log(x_c) \tag{4}$$

There are several types of the autoencoder, including Vanilla, Deep and Regularized Autoencoder [5].

- Vanilla Autoencoder (VA) is the simplest autoencoder. It is a neural network with one hidden layer H. H has less dimensions as compared to the input layer. Vanilla Autoencoder learns how to reconstruct the input using Adam optimizer and the mean squared error loss function.
- Deep Autoencoder (DA) is the extension of Vanilla version with three fully connected hidden layers. This architecture is presented in Figure 2 . The DA can be trained with limited training samples. It is also able to capture more abstract representation from the data in order to improve the classification performance.
- Regularized Autoencoder (RA) encourages the model to have other properties, such as rank deficiency and sparsity apart from the ability of reconstructing the input **x**. The RA can be either spare or denoising autoencoder. Spare autoencoder involves a sparsity penalty Ω(H) in the core layer H, while the denoising autoencoder changes the reconstruction error term of the cost function to generalize cases that are not presented in the training dataset.

In this paper, the RA is also combined with a deep architecture to form the so-called Regularized Deep Autoencoder (RDA). The construction of the autoencoder is based on samples from the legitimate data class [12]. Its motivation is to learn an embedded low

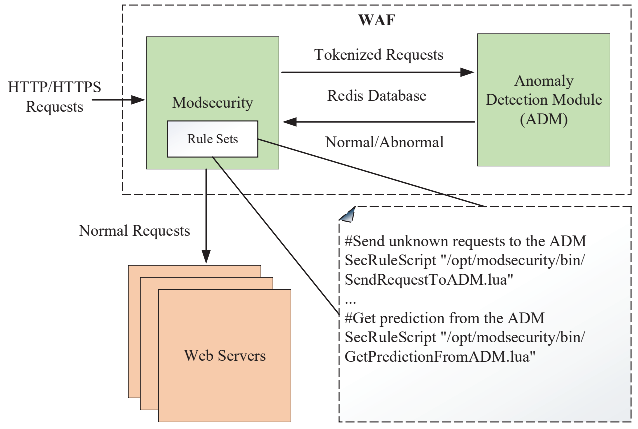
WAF
HTTP/HTTPS Requests
Modsecurity
Rule Sets
Tokenized Requests
Redis Database
Normal/Abnormal
Anomaly Detection Module (ADM)
Normal Requests
Web Servers
#Send unknown requests to the ADM
SecRuleScript "/opt/modsecurity/bin/SendRequestToADM.lua"
...
#Get prediction from the ADM
SecRuleScript "/opt/modsecurity/bin/GetPredictionFromADM.lua"

Figure 3: Web Application Firewall based on the combination of the RDA and Modsecurity.

dimensional subspace that can represent the normal requests with minimal reconstruction error.

2.3 Selecting the appropriate threshold

The detection module receives the HTTP/HTTPS request as an input and determines if the request is malicious or not. This process is often complex since the complicated interactions, such as co-occurrence and others of method called are all involved in the decision making. These complexities make the autoencoder an ideal candidate for eliminate web attacks. In the autoencoder, it is essential to select the appropriate threshold, which triggers the trade-off between True Positive (TPR) and False Positive Rate (FPR). Generally, the threshold θ can be computed based on the average reconstruction error $\bar{E}$ and standard deviation s as.

$$\bar{E} = \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x} - \mathbf{x'}\| = \frac{1}{N}\sum_{i=1}^{N}(\sum_{j=1}^{n}(x_j - x'_j)^2) \tag{5}$$

$$s = \sqrt{\frac{\sum_{i=1}^{N}(E_i - \bar{E})^2}{N-1}} \tag{6}$$

$$\theta = \bar{E} + \alpha \times s \tag{7}$$

where N and n are the number and dimension of the input vectors, used to validate the model. α is a trade-off parameter that is selected depending on the training dataset and the classification problem at hand. In this paper, α is set equal to 3. The standard deviation serves as a measure of uncertainty. For a normal distribution, three standard deviations include 99.73% of the samples. We assume that there are several outliers in the training dataset. This α value allows us to eliminate the outliers and obtain a compact and reliable description around the legitimate class. Given the predefined threshold θ, a request is considered as malicious when its reconstruction error is larger than θ.

2.4 The overall architecture

In the present work, the autoencoder is also deployed in a WAF to detect and eliminate web attacks. We have selected Modsecurity, an open-source WAF that has evolved to provide the filtering capabilities along with other security features in a wide range of platforms, including NGINX, Microsoft IIS and Apache HTTP Server [15]. As can be seen in Figure 3, Modsecurity is utilized to preprocess the input HTTP request and produce a tokenized vector. Such vector is pushed into the reliable REDIS queue, who acts as a messaging server to implement processing the different messaging tasks. The role of the Anomaly Detection Module (ADM) is to determine whether a request is malicious. It then responses to Modsecurity, which is equipped with a mechanism to block unwanted HTTP queries before they can enter the web application. The communication between the ADM and Modsecurity is achieved via two different Lua APIs, namely SendRequestToADM.lua and GetPredictionFromADM.lua. The Lua APIs are lightweight and can be executed internally as part of the web server.

3 EXPERIMENTS

In this section, we evaluate the original Vanilla Autoencoder and its variants and compare these algorithms with other one-class methods, whose classification model is constructed using data solely from the legitimate class. We also assess the architecture, where the autoencoder is combined with Modsecurity to protect a website in real time. All the codes in the present work were written using Keras [2] and Scikit-Learn [13] libraries. They were executed on a PC, running Ubuntu 16.04 with Intel Core i5 and 8 GB RAM.

3.1 Data specification

The autoencoder neural networks are evaluated and benchmarked on the CSIC 2010 dataset. This dataset is divided into two subsets. The first subset contains 36,000 normal request that can be used to train the unsupervised learning methods. The second subset contains 36,000 normal and more than 25,000 abnormal requests, that can serve for testing purpose. The web attacks in CSIC 2010 involve SQL injection, XSS, CRLF injection, buffer overflows and many others.

In addition, we evaluate the different algorithms in a practical web application, which was developed on the WordPress framework. Particularly, we execute a number of web attacks using the Burp Suite Professional [19], a powerful web scanner that covers over 100 generic vulnerabilities, including SQL injection, XSS and other OWASP top 10 vulnerabilities [11]. The queries are then manually labeled based on the expert knowledge. There is a total of 16,742 normal and 14,824 abnormal requests in the collection.

3.2 Evaluation measures

The evaluation metrics that are used in the present work include Precision, Recall, F1-score and Area Under the ROC Curve (AUC). These are computed using True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP and TN are the number of malicious and legitimate HTTP/HTTPS requests that are correctly classified. FP is the number of normal requests that are incorrectly classified as malicious, while FN is the number of abnormal requests that are incorrectly classified as legitimate. We have that

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

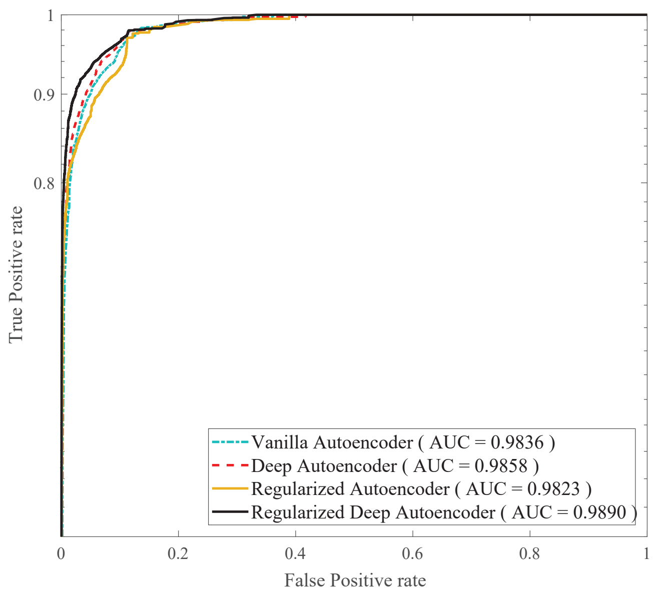

Figure 4: Comparing the various autoencoder variants, including Vanilla, Deep, Regularized and Regularized Deep Autoencoder on the CSIC 2010 dataset.

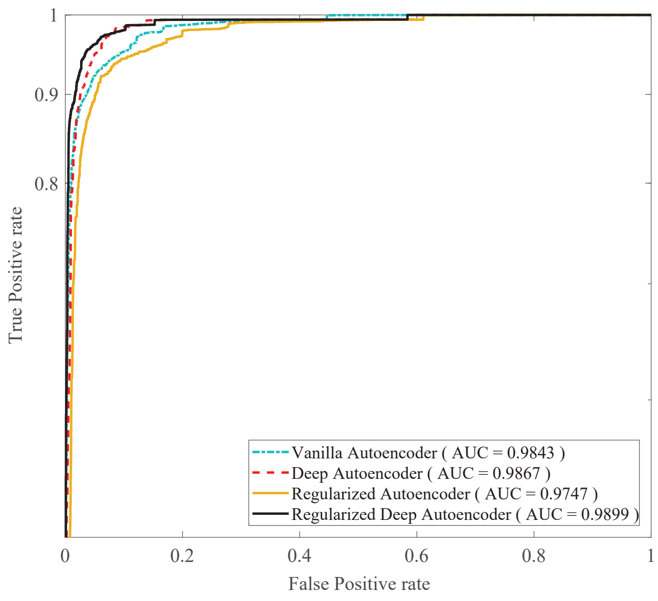

Figure 5: Comparing the various autoencoder variants, including Vanilla, Deep, Regularized and Regularized Deep Autoencoder on the real-world dataset.

$$Recall = \frac{TP}{TP + FN} \tag{9}$$

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{10}$$

Table 1: The precision, recall and F1-score, related to the RDA and other methods on the CSIC 2010 dataset

Method	Precision	Recall	F1-score
Modsecurity + CRS (PL = 1)	1.0000	0.6517	0.7890
Modsecurity + CRS (PL = 2)	0.9368	0.6853	0.7915
Modsecurity + CRS (PL = 3)	0.8414	0.7452	0.7904
Modsecurity + CRS (PL = 4)	0.6820	0.7908	0.7321
One-class SVM + 30 features [10]	0.5963	0.5871	0.5916
Stacked Autoencoder + Isolation Forest [17]	0.8029	0.8832	0.8412
Regularized Deep Autoencoder	0.9464	0.9462	0.9463
LSTM network [18]	0.9770	0.9790	0.9780

Table 2: The precision, recall and F1-score, related to the RDA and other methods on the real-world dataset

Method	Precision	Recall	F1-score
Modsecurity + CRS (PL = 1)	1.0000	0.5246	0.6881
Modsecurity + CRS (PL = 2)	0.9981	0.5347	0.6963
Modsecurity + CRS (PL = 3)	0.9806	0.5745	0.7245
Modsecurity + CRS (PL = 4)	0.9678	0.6036	0.7434
Regularized Deep Autoencoder	0.9533	0.9528	0.9530
LSTM network [18]	0.9860	0.9859	0.9859

F1-score is the harmonic mean of the Precision and Recall. AUC provides a single measure that can be interpreted as the average True Positive Rate (TPR) over the entire range of possible False Positive Rate (FPR). The computational complexity is also considered in this work. It is measured by the time that the web server requires to process a single request.

3.3 Results

Figure 4 illustrates the performance of the autoencoder variants on the CSIC 2010 dataset. The intuitive motivation to select the best candidate that can be deployed in combination with Modsecurity. It is observed that the original Regularized Autoencoder achieves the lowest AUC (0.9823). The Regularized Deep Autoencoder (RDA) (0.9890 AUC) is proved to outperform the Vanilla Autoencoder (0.9836 AUC) and the Deep Autoencoder (0.9858 AUC). Although the gap is relatively small, the better TPR is achieved at all the FPR values. Similar observations can be obtained on the real-world dataset (see Figure 5 for more detail). Based on these observations, we have selected the RDA and implemented it with Modsecurity to detect malicious patterns.

In Tables 1, 2, we compare the RDA with other supervised learning methods, including the Stacked Autoencoder and Isolation Forest in [17]. The OWASP Modsecurity Core Rule Set (CRS) is a set of signatures, which is able to protect web application from a wide variety of attacks. Modsecurity Core Rule Set can trigger action using the anomaly scores corresponding to each HTTP request. The action can be either passing or dropping the request. The CRS provides the Paranoia Level (PL) setting that allows us to select the desired level of rule checks. As the PL increases, the CRS introduces additional rules, giving a high level of protection [15]. It is obvious that an increase in the PL can lead to an increase in the Recall and a decrease in the Precision. Table 1 also illustrates the accuracy of the one-class SVM when trained using 30 different features. The features are directly extracted from the HTTP queries based on the work of Nguyen et al. [10]. One-class SVM has been proved to outperform other conventional one-class methods in various application domains. This observation implies that the hand-crafted features are not a reliable source of information to train the unsupervised learning methods in order to detect web attacks.

The RDA is demonstrated to be inferior to the Long Short-Term Memory (LSTM) network [18]. It has to be noted that the LSTM network is discriminative. Model that is trained using data from both malicious and legitimate classes is definitely better that the model that is trained using data from one side. The LSTM however requires the requests to be labeled, which is not always possible in practice. In the same configuration setting, the RDA is much better than the stacked autoencoder in [17]. In the RDA, we preprocess the requests, and thus, extracting essential information to build the classification model. Decision is directly rendered using the reconstruction error rather than the isolation forest.

In Table 3, we assess the RDA ability to detect common web application security vulnerabilities on the real-world dataset. It can be highlighted that this algorithm is able to recognize most of the injection attacks, while the accuracy on XSS is approximately 96%. Clearly, the RDA provides a viable option to eliminate the malicious HTTP requests.

In Table 4, the RDA is coupled with Modsecurity to protect a website in real time. In general, it is important to provide a balance between security and the system performance. The extra second the web application firewall spends on scanning is an extra second the user must wait for the response. In order to validate the above combination, we have run a script to execute 31,566 requests with 10 concurrent threats, which are equivalent to 10 users accessing to the web server at the same time. From Table 4, we observe that the RDA is comparable to Apache (without WAF) in terms of average computation time. It should be noted that Apache is not equipped with blocking option and must handle all the incoming requests. Modsecurity has a slightly better performance with respect to the RDA and requires only 4.9 milliseconds to process a single request. The LSTM is computationally expensive. Hence, it cannot be deployed in the practical applications.

Table 3: The RDA performance on the common web application security vulnerabilities, which is evaluated using the real-world dataset

Attacks	# of Requests	# of Detected Request	Recall
SQL Injection	7,858	7,472	0.9509
Cross-site Scripting(XSS)	1,112	1,064	0.9604
XPath Injection	2,329	2,066	0.8871
Local File Inclusion(LFI)	663	535	0.8069
Sever-side Template Injection	1,053	996	0.9459
Code Injection	808	688	0.8515
OS command Injection	81	81	1.0000
Server side Request Forgery	205	205	1.0000
Others	715	652	0.9118

Table 4: The computation time of RDA, when combined with Modsecurity and other methods

Method	Computation time
Apache	5.1 ms/request
Modsecurity + CRS	4.9 ms/request
Modsecurity + RDA	5.1 ms/request
LSTM network [18]	23.8 ms/request

4 CONCLUSIONS

This paper proposes the use of autoencoder in detecting web attacks. The autoencoder receives in input a tokenized request. It then classifies whether the request is deemed to be malicious. Experiments on the CSIC 2010 dataset demonstrate that the Regularized Deep Autoencoder (RDA) outperforms the other autoencoder variants. This algorithm is able to achieve a better performance with respect to the one-class SVM that is trained using the hand-crafted attributes. We also developed an architecture, where the RDA is coupled with the well-known Modsecurity to protect a website. Such website is based on the WordPress framework. The RDA is proved to be comparable to the original Modsecurity in terms of computation time and hence, being amenable to immediate applications.

ACKNOWLEDGMENTS

This research is funded by Hanoi University of Science and Technology (HUST) under grant number T2017-PC-162.

REFERENCES

[1] Guillaume Alain and Yoshua Bengio. 2014. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research* 15, 1 (2014), 3563–3593.

[2] François Chollet et al. 2015. Keras.

[3] Symantec Corproration. 2016. Internet security threat report.

[4] Carmen Torrano Giménez, Alejandro Pérez Villegas, and Gonzalo Álvarez Marañón. 2010. HTTP data set CSIC 2010.

[5] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[6] Kenneth L Ingham and Hajime Inoue. 2007. Comparing anomaly detection techniques for http. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 42–62.

[7] Rafał Kozik, Michał Choraś, Rafał Renk, and Witold Hołubowicz. 2015. Patterns extraction method for anomaly detection in HTTP traffic. In *International Joint Conference*. Springer, 227–236.

[8] Christopher Kruegel, Giovanni Vigna, and William Robertson. 2005. A multi-model approach to the detection of web-based attacks. *Computer Networks* 48, 5 (2005), 717–738.

[9] Jingxi Liang, Wen Zhao, and Wei Ye. 2017. Anomaly-Based Web Attack Detection: A Deep Learning Approach. In *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*. ACM, 80–85.

[10] Hai Thanh Nguyen, Carmen Torrano-Gimenez, Gonzalo Alvarez, Slobodan Petrović, and Katrin Franke. 2011. Application of the generic feature selection measure in detection of web attacks. In *Computational Intelligence in Security for Information Systems*. Springer, 25–32.

[11] Top OWASP. 10. Application Security Risks-2017. Open Web Application Security Project (OWASP).

[12] Yao Pan, Fangzhou Sun, Jules White, Douglas C Schmidt, Jacob Staples, and Lee Krause. 2018. Detecting Web Attacks with End-to-End Deep Learning. (2018).

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[14] Konrad Rieck and Pavel Laskov. 2006. Detecting unknown network attacks using language models. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 74–90.

[15] Trustwave Spiderlabs. [n. d.]. Modsecurity 3.0 - Open Source Web Application Firewall. Retrieved September 2, 2018 from http://www.modsecurity.org/

[16] Carmen Torrano-Gimenez, Hai Thanh Nguyen, Gonzalo Alvarez, and Katrin Franke. 2015. Combining expert knowledge with automatic feature extraction for reliable web attack detection. *Security and Communication Networks* 8, 16 (2015), 2750–2767.

[17] Ali Moradi Vartouni, Saeed Sedighian Kashi, and Mohammad Teshnehlab. 2018. An anomaly detection method to detect web attacks using Stacked Auto-Encoder. In *Fuzzy and Intelligent Systems (CFIS), 2018 6th Iranian Joint Congress on*. IEEE, 131–134.

[18] Jiabao Wang, Zhenji Zhou, and Jun Chen. 2018. Evaluating CNN and LSTM for Web Attack Detection. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*. ACM, 283–287.

[19] Portwigger web security. [n. d.]. Burp Suite Professional. Retrieved September 2, 2018 from https://portswigger.net/burp

[20] Guiqin Yuan, Bo Li, Yiyang Yao, and Simin Zhang. 2017. A deep learning enabled subspace spectral ensemble clustering approach for web anomaly detection. In *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 3896–3903.