Web Application Attacks Detection Using Deep Learning ⋆

Nicolás Montes, Gustavo Betarte[1], Rodrigo Martínez[1], and Alvaro Pardo[2]

[1] Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay
[2] Departamento de Ingeniería, Facultad de Ingeniería y Tecnologías
Universidad Católica del Uruguay, Uruguay

Abstract. This work investigates the use of deep learning techniques to improve the performance of web application firewalls (WAFs), systems that are used to detect and prevent attacks to web applications. Typically, a WAF inspects the HTTP requests that are exchanged between client and server to spot attacks and block potential threats. We model the problem as a one-class supervised case and build a *feature extractor* using deep learning techniques. We treat the HTTP requests as text and train a deep language model with a transformer encoder architecture which is a self-attention based neural network. The use of pre-trained language models has yielded significant improvements on a diverse set of NLP tasks because they are capable of doing transfer learning. We use the pre-trained model as a *feature extractor* to map a HTTP request into a feature vector. These vectors are then used to train a one-class classifier. We also use a performance metric to automatically define an operational point for the one-class model. The experimental results show that the proposed approach outperforms the ones of the classic rule-based ModSecurity configured with a vanilla OWASP CRS and does not require the participation of a security expert to define the features.

Keywords: Web Application Firewall, Anomaly Detection, Deep Learning

1 Introduction

It has become a regular security practice to deploy a Web Application Firewall (WAF) [9] to identify attacks that exploit vulnerabilities of web applications. A WAF is a piece of software that intercepts and inspects all the traffic between the web server and its clients, searching for attacks inside the HTTP packet contents. An implementation of an open source WAF that has become a *de facto* standard is ModSecurity [24]. The actions this WAF undertakes are driven by rules that specify, by means of regular expressions, the contents of the HTTP packets to

⋆ This research was partially supported by a grant given to Nicolás Montes from ANII (http://anii.org.uy) and was done in the context of projects FMV_1_2017_136337 (Fondo María Viñas, ANII) and WAFINTL from ICT4V center (http://ict4v.org).

be analyzed and eventually flagged as potential attacks. ModSecurity comes equipped with a default set of rules, known as the owasp Core Rule Set (owasp crs) [15], for handling the most usual vulnerabilities included in the OWASP Top Ten [16]. However, this rule-based approach has some drawbacks: rules are static and rigid by nature, so the owasp crs usually produces a rather high rate of false positives, which in some cases may be close to 40% [8] that would potentially lead to a denial of service of the application. The systematic review presented in [22] analyzes the available scientific literature focused on detecting web attacks using machine learning techniques. In [4,3,13] we have presented solutions where the rule-based detection approach of ModSecurity is complemented with machine learning-based models to mitigate the rule-based approach's drawbacks.

In this work we present an approach that makes use of deep learning techniques to improve the performance of ModSecurity. It consists of a two step learning framework: first we build a *feature extractor* using deep learning techniques; then we train a one-class supervised model. We treat the http requests as raw text and pre-train a deep language model with the architecture proposed in [12]. These models can operate in huge amounts of text and are called self-supervised because the optimization of the network does not require labels (we will explain this in Section 3).

The structure of the rest of the paper is as follows: Section 2 describes the deep learning techniques and the related work. Section 3 presents the deep learning framework. The outcomes are described and discussed in Section 4. Further work and conclusions are presented in Section 5.

2 Background and related work

NLP techniques have been greatly improved by the advancements of deep learning [12,6,19,17]. These models rely on a two-step approach. First, they learn deep contextual word representation from raw text in a self-supervised way (stage referred as pre-training). Then, this pre-trained language model can be applied to downstream NLP tasks by choosing between two learning strategies: *feature-based* and *fine-tuning*.

Traditional NLP techniques represent words as atomic units and text is transformed into a numeric vector using one-hot encoding. There are two main problems with this approach. First, there is no notion of similarity between words, as they are represented as indices in a vocabulary [14]. Additionally, the size of the vector is as large as the size of the vocabulary, $|V|$, making machine learning methods prone to problems related with high dimensional feature spaces such as the curse of dimensionality. With the progress of machine learning techniques it has become possible to train more complex models. Probably one of the most successful concept is to use distributed representations of words, also know as *word embedding*. In this approach words are represented in a continuous vector space with much lower dimension than $|V|$. Additionally, it has been shown that words with semantic similarities tend to be nearby in the vector space [2]. In the last decade, word embeddings have established themselves as a core element

of many NLP systems. However, as word embedding techniques are static they miss a crucial element for fully capturing local contexts, that is, the semantic and syntactic meaning of words. These methods actually learn to capture the general (most common) context of words in their representations, but they are not able to handle polysemy. Replacing static embeddings with deep contextualized representations has yielded significant improvements on a diverse set of NLP tasks. The idea is simple, a word is assigned a representation that is a function of the entire input sequence (the whole text sequence). The success of deep contextualized word representations suggests that despite being trained with only a language modelling goal, they learn highly transferable and task-agnostic properties of the language [7].

In this work we propose the use of deep contextualized representation of HTTP requests to extract feature vectors that then will be used to train a classifier to detect attacks to web applications. In a first step we create a deep pre-trained language model from scratch using a set of HTTP requests from the web application that we aim to protect. In a second step, we use the *feature-based* strategy to transform each HTTP request into a feature vector. That is, once we have obtained the pre-trained model, we convert each HTTP request into a numeric representation using the weights of the last layer of the network, also known as *feature extraction*. With these representations as input we build a One-Class Classification model (OCC).

Related work In [10] Kruegel and Vigna propose an anomaly detection approach where they model specific characteristics of the URL parameters, such as parameter length and input order to generate a probabilistic grammar of each parameter. In our one-class approach we work using the whole request, not only the URL parameters, capturing the normal behavior by modeling the occurrence of a specific set of tokens. This allows us to capture the behavior of the data sent in the normal use of the application. In our approach we also deal with attacks present in the body and header of the requests.

Several authors propose anomaly detection techniques that work over simplification of the application's parameter values. In [5], numbers and alphanumeric sequences are abstracted away, representing each category with a single symbol. In [23] Torrano-Giménez et al present an anomaly detection technique that instead of using the tokens themselves uses a simplification that only considers the frequencies of three sets of symbols: characters, numbers and special symbol. In our approach the whole request is analyzed without any further simplification.

The work presented in [28] uses word embeddings to represent the URLs. This approach has three steps. First, an ensemble clustering model is applied to separate anomalies from normal samples. Then they use *word2vec* to get the semantic representations of anomalies. Finally, another multi-clustering approach clusters anomalies into specific types. In our model, static embeddings (word2vec) are replaced with deep contextualized representations. We use these representations to get the semantic representations of normal data and use it as input to build the one-class model. In [27] Yu et al propose a method that uses Bidirectional Long Short-Term Memory (Bi-LSTM) with an attention mecha-

nism to model the HTTP traffic. This approach is supervised, as they train the Bi-LSTM network to predict whether a request is anomalous or not.

In [18] Qin et al propose a model which learns the semantics of malicious segments in payload using a Recurrent Neural Network (RNN) with an attention mechanism. The payload is transformed into a hidden state sequence by a RNN and then an *attention mechanism* is used to weight the hidden states as the feature vector for further detection. Thus, they also can use the hidden state of the network as features for a second classifier. The difference with the learning technique that we propose is that they learn the weights of the RNN, the feature extractor model, using normal and abnormal instances. In our case, we build a self-supervised pre-trained model using only normal data.

The work [25] proposes a model that uses a stacked auto-encoder (SAE) and a deep belief network as feature learning methods in which only normal data is used in the learning phase. Subsequently, OCSVM, Isolation Forest and Elliptic Envelope are used as classifiers. In this work features of the HTTP are extracted using n-grams and then deep learning models are applied to reduce the dimensionality generated by the n-grams vectors. In our case we work directly with the HTTP request and avoid building the n-grams which require large amounts of data to correctly capture the statistics of each modelled field.

3 A two-step learning approach for anomaly detection

We propose a learning architecture composed of a two-step method to improve web application anomaly detection models. In a first step, we create a deep pre-trained language model using only normal HTTP requests to the web application. In a second step, we use this model as a feature extractor and train a one-class classifier. That is, each web application has its own model (both the pre-trained language model and the one-class classifier). In the following sections we describe the components of the proposed learning architecture depicted in Figure 1.

3.1 Pre-training a HTTP language model

We train a language model for the HTTP requests in a self-supervised way. We use a Robustly Optimization Bidirectional Encoder Representations from Transformers architecture (RoBERTa) [12]. Using this model each HTTP request is transformed into a numeric vector that captures the contextual information of each token present in the request. The architecture of the network used to build the language model is a multi-layer bidirectional Transformer Encoder [26]. This is an attention-based architecture for modeling sequential data which is an alternative to recurrent neural networks (RNN) and is capable of capturing long range dependencies in sequential data.

The proposed model (see top Figure 1), is composed of a stack of **L** identical transformer encoder layers, as detailed at the bottom of Figure 1. Each encoder layer contains two types of sub-layers. The first one is a multi-head self-attention mechanism, which helps looking at other tokens in the sequence while encoding

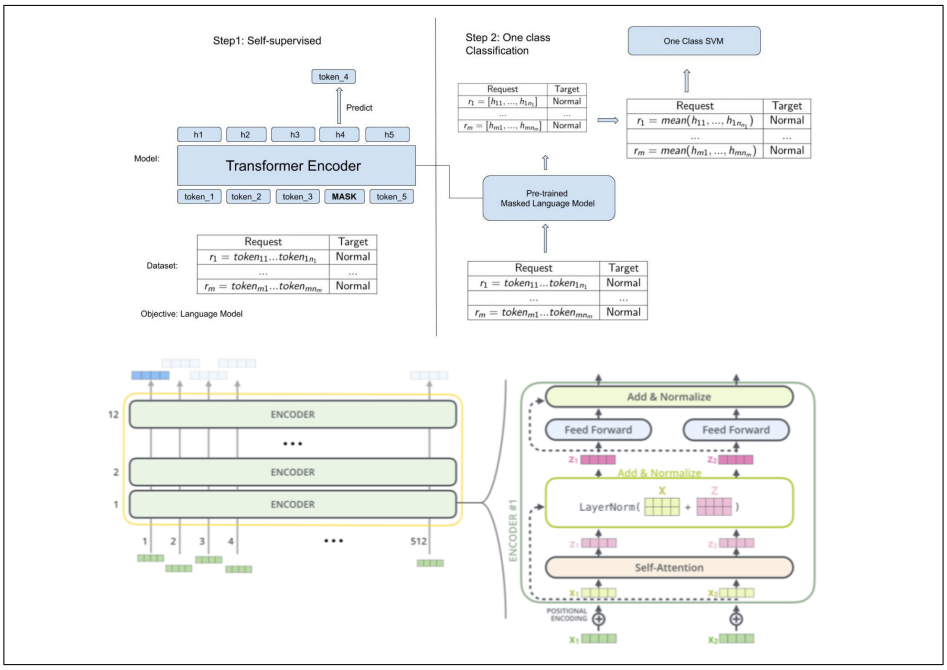

Fig. 1: Top: Proposed architecture. Left: transformer encoder used to extract the contextual representation of each token $token_{ij}$ in the request r_i. Right: each request r_i is represented as the mean of token deep contextualized representation h_{ij} and how they are used to train a one-class classifier. Bottom: Architecture of the Transformer Encoder (Jay Alammar, 2018 [1])

a specific token. The second sub-layer is simply a feed-forward network (FFN), which is applied to each position (token representation from previous layer) separately and identically. Because our implementation is almost identical to the original, for a detailed description of the model architecture, we refer readers to [26,12]. We denote the number transformer encoder blocks as **L** (see top Figure 1), the hidden size as **H** (the output of the transformer encoder denoted as h in Figure 1), and the number of self attention heads as **A**. Our model uses the following set of parameters (L=12, H=768, A=12, Total Parameters = 125M).

Token encoding and model training. The input to the model composed of L blocks of transformers is a tokenized version of the HTTP request. For that we use a Byte-Pair Encoding (BPE)[21] tokenizer, a hybrid between character and token-level representations. It relies on sub-word units which are extracted by performing statistical analysis of a training corpus. We use the same tokenizer learned by [19][3], a clever implementation of BPE that uses *bytes* instead

[3] We could have chosen another pre-trained BPE tokenizer instead of the one proposed in [19]. The key point is to use a BPE tokenizer trained on huge corpus (40 GB of text) because they can tokenize any word (and any character) of any language without using the *unknown* token.

of unicode characters as the basic sub-words units. This tokenizer has a sub-word vocabulary of 50K units that can still encode any input string without introducing any *unknown* tokens.

Given the model architecture the next step is to define the training strategy, that is, the learning goal and the training mechanisms. In our case, in order to learn the deep contextualized representation of tokens we apply a self-supervised learning approach. We randomly masks some of the tokens from the input, and then the goal is to predict the original masked token based only on its context. In [6] they refer to this procedure as *Masked Language Model* (MLM). In contrast to denoising auto-encoders, these models only predict the masked words rather than reconstructing the entire input [6]. In an attempt to predict the masked tokens, the model should be able to extract some information from the language, not only structural information but some semantic information as well. This information is encoded in the weights of the encoding layers.

3.2 One-Class Classification

The pre-trained model detailed in section 3.1 takes a request r_i and tokenizes it to obtain a representation $r_i = \{token_{i1}, ..., token_{in_i}\}$, where n_i is the number of tokens in the request r_i (the model has a *max length* of 2048 tokens as inputs to be processed). Then, generates as output a deep contextual representation of each token. This deep representation is obtained using the weights of the encoder's last layer. In this way, each request r_i is transformed into a numeric vector $\{h_{i1}, ..., h_{in_i}\}$. Each $h_{ij} \in R^H$ is the vector representation for the $token_{ij}$ in r_i (where H = 768 is the size of the encoder hidden layer).

In order to get a representation of the full request, the most common technique is to average token representations to produce a vector $\bar{r}_i \in R^H$ such that: $\bar{r}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} h_{ij}$. With these procedure we transform a set of normal HTTP requests $D = \{r_1, .., r_m\}$ into a numeric form $\bar{D} = \{\bar{r}_1, .., \bar{r}_m\}$, each $\bar{r}_i \in R^H$. We perform a One-Class Classification model (OCC), with a One-Class Support Vector Machine (OCSVM), with these representations. We operate in a scenario in which only valid requests are known, and no requests tagged as attacks are necessary. We believe that is a realistic approach, where valid traffic could be collected, for instance, as the result of performing functional testing of the application.

Once we have the feature vector mentioned above, we apply the well-know OCSVM classifier introduced in [20] with a Radial Basis Function (RBF) Kernel. They develop an algorithm that returns a function f that takes the value $+1$ in a "small" region capturing most of the training data points and -1 elsewhere. The strategy is to map the data into the feature space corresponding to the kernel and to separate them from the origin with maximum margin. For a new point x, the value $f(x)$ is determined by evaluating which side of the hyper-plane it falls on in feature space. For each sample, we can calculate the signed distance to the separating hyper-plane. The distance is positive for an inlier (considered as normal) and negative for an outlier (a possible attack). We can set a threshold (θ)

and classify a sample as normal if the distance to the hyper-plane is greater than θ. Varying θ between -1 and $+1$ we obtain a ROC with different operational points. Below we will explain how to automatically obtain the best θ using a grid search approach.

Estimation of the optimal operational point. We must set up two parameters to optimize the performance of the OCSVM: ν and γ. γ parameter is required by the RBF kernel to define a frontier. ν corresponds to the probability of finding a new, but normal, observation outside the frontier. To find the optimal parameters we use a traditional grid-search method. In the case of supervised classification, we can use performance metrics such as F-score or the overall accuracy to evaluate each configuration of parameters. However, these metrics rely on positive and negative samples so it is not possible to use them in an anomaly detection scenario. Nevertheless, [11] introduces a performance measure, $\hat{F}$, that can be estimated from normal and unlabeled examples. They show that $\hat{F}$ is proportional to the square of the geometric mean of precision and recall. Thus, they argue that has roughly the same behaviour as the F_1-score (the harmonic mean of the precision and recall). Therefore, we use a grid-search with the $\hat{F}$ metric for selecting the best parameters of the OCSVM classifier. We use a validation set with only normal and unlabeled examples. The best parameters for both datasets used in Section 4 are: $\nu = 0.05$ and $\gamma = 0.5$

4 Results

The performance of the proposed method is analyzed in terms of True Positive Rate (TPR) and False Positive Rate (FPR). In our case, TPR and FPR indicate the ratio of requests correctly and incorrectly classified as attacks, respectively. To evaluate the proposed method we shall use the same datasets used in our previous work [3,13]. The CSIC2010 dataset embodies a collection of normal and abnormal HTTP requests for a web application that provides functionalities to perform an on-line shopping. The dataset contains 36.000 valid request for training, other 36.000 for testing and 25.000 request of anomalous traffic. We use the DRUPAL dataset in order to evaluate the model on a real life application. We crafted this dataset by registering three days of incoming traffic to the public website of a University. The dataset contains 65.000 valid request and 1.287 real attacks.

One of the main goals of this work is to reduce the amount of false positives generated by ModSecurity without decreasing the TPR. For this reason we compare our results against ModSecurity configured with the OWASP CRS version 3 out of the box with two paranoia levels. We also compare the results with the classic information retrieval approach based one-class model presented in [3] later improved in [13]. In this case, a security expert defines the features to be extracted and used to train a one-class classifier based on a Gaussian mixture model. One objective of this work is to compare the features automatically extracted with deep learning with the ones selected by the expert.

Table 1. TPR and FPR for each dataset. (*) One-class classifier using features manually selected by an expert (HTTP tokens). The operational point was manually selected by the authors. (+) The operational point was automatically selected (see Section 3.2).

	DRUPAL		CSIC 2010	
Method	**TPR**	**FPR**	**TPR**	**FPR**
ModSecurity OWASP CRS v3 -PL 1	29.55%	15.57%	26.62%	0.00%
ModSecurity OWASP CRS v3 -PL 2	77.89%	49.93%	29.48%	0.00%
One-class classifier from [13] (*)	94.43%	6.00%	39.63%	5.37%
RoBERTa + OCSVM (+)	95.00%	3.73%	47.10%	7.54%

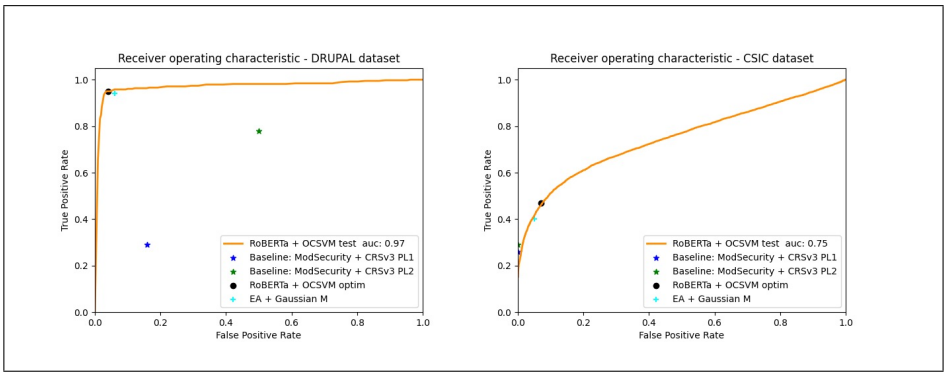

Fig. 2: One-class ROC curve varying the θ value

The evaluation was performed, on each of the datasets, using 70% of the valid requests for training and the rest of the dataset (30% of valid and 100% of attacks) for testing. The results of our proposal (RoBERTa + OCSVM) in Table 1 were obtained with parameters ν, γ and θ found automatically as explained in Section 3. In Figure 2 we present the results in terms of a ROC curve (constructed in terms of TPR and FPR) The solid line represents the different operation points of the OCSVM model. The stars represents the performance of ModSecurity. The plus symbol represents the operating point achieved by the EA+Gaussian Mixture model in [13]. The circle represents the operating point achieved by OCSVM with the estimated ν, γ and θ as explained above.

In the case of the DRUPAL dataset the ROC curve shows that there are several points that outperform all configurations of ModSecurity. If we compare the results with our baseline, the best configuration ModSecurity detects 75% of the attacks, whereas RoBERTa + OCSVM detects 95.00%. The FPR of ModSecurity is 39.69% and RoBERTa + OCSVM is 3.37%. As to the dataset CSIC2010 the TPR is higher than all versions of ModSecurity. CSIC2010 is a synthetic dataset constructed adding some attacks and anomalous requests to a set of normal ones. ModSecurity with paranoia levels 1 and 2 does not produce any false positives at the expense of extremely low TPR. With a more strict paranoia level (PL 3) FPR is 13.95% and TPR is 52.61%. Our method produces a low FPR (7.54%) with a similar TPR (47.10%) for this dataset.

5 Conclusion and further work

To the best of our knowledge, the method we propose is a first attempt in using a deep transformer based language representation of HTTP requests to address the problem of web applications attack detection.

We used two different datasets to pre-train a deep language model for the HTTP requests without requiring a security expert to define the set of features. We have proposed a two-step learning approach consisting in first mapping a HTTP request into a continuous space using a transformer encoder and then applying a OCSVM to discriminate normal traffic from attacks. We have used a performance metric proposed by [11] to automatically obtain the parameters of the OCSVM.

We find that the results presented in Section 4 are quite promising. They outperform ModSecurity using the most widely adopted rules and are slightly better than those presented in [3] later improved in [13] with the advantage of not requiring the participation of a security expert to define the features.

As future work, we intend to re-train the pre-trained language model with the dataset DRUPAL with more HTTP requests and check whether we can improve the language model. We also plan to use a set of attacks and explore the *fine-tuning* approach. That strategy consists in adding one additional layer and *fine-tuning* all the parameters of the whole network. The goal of this supervised downstream task is to check whether the model generalizes.

References

1. The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. `http://jalammar.github.io/illustrated-transformer/`. (Accessed on 02/14/2021).
2. Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.
3. G. Betarte, E. Giménez, R. Martinez, and Á. Pardo. Improving web application firewalls through anomaly detection. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 779–784. IEEE, 2018.
4. G. Betarte, R. Martínez, and Á. Pardo. Web application attacks detection using machine learning techniques. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1065–1072. IEEE, 2018.
5. I. Corona, D. Ariu, and G. Giacinto. Hmm-web: A framework for the detection of attacks against web applications. In *Proceedings of ICC 2009*, pages 1–6, 2009.
6. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
7. K. Ethayarajh. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings. *arXiv preprint arXiv:1909.00512*, 2019.
8. C. Folini. Handling false positives with the owasp modsecurity core rule set, 2016.
9. A. J. Hacker. Importance of web application firewall technology for protecting web-based resources. *ICSA Labs an Independent Verizon Business*, 2008.

10. C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of CCS 2003*, pages 251–261. ACM, 2003.
11. W. S. Lee and B. Liu. Learning with positive and unlabeled examples using weighted logistic regression. In *ICML*, volume 3, pages 448–455, 2003.
12. Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
13. R. Martínez. Enhancing web application attack detection using machine learning. Master's thesis, Facultad de Ingeniería,UdelaR - Área Informática del Pedeciba, Uruguay, 2019.
14. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
15. OWASP. Owasp modsecurity core rule set project. URL: `https://coreruleset.org`. Last visited on 14/02/2021.
16. OWASP. Owasp top ten project. URL: `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`. Last visited on 14/02/2021.
17. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.
18. Z.-Q. Qin, X.-K. Ma, and Y.-J. Wang. Attentional payload anomaly detector for web applications. In *International Conference on Neural Information Processing*, pages 588–599. Springer, 2018.
19. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
20. B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.
21. R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.
22. T. Sureda Riera, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-J. Martínez Herraiz, and J.-A. Sicilia Montalvo. Prevention and fighting against web attacks through anomaly detection technology. a systematic review. *Sustainability*, 12(12), 2020.
23. C. Torrano-Gimenez, A. Perez-Villegas, G. Á. Marañón, et al. An anomaly-based approach for intrusion detection in web traffic. *Journal of Information Assurance and Security*, 5(4):446–454, 2010.
24. I. Trustwave Holdings. Modsecurity: Open source web application firewall.
25. A. M. Vartouni, M. Teshnehlab, and S. S. Kashi. Leveraging deep neural networks for anomaly-based web application firewall. *IET Information Security*, 13(4):352–361, 2019.
26. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.
27. Y. Yu, H. Yan, H. Guan, and H. Zhou. Deephttp: semantics-structure model with attention for anomalous http traffic detection and pattern mining. *arXiv preprint arXiv:1810.12751*, 2018.
28. G. Yuan, B. Li, Y. Yao, and S. Zhang. A deep learning enabled subspace spectral ensemble clustering approach for web anomaly detection. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3896–3903. IEEE, 2017.