Hindawi
Security and Communication Networks
Volume 2022, Article ID 5280158, 14 pages
https://doi.org/10.1155/2022/5280158

Research Article

Web Application Firewall Using Machine Learning and Features Engineering

Aref Shaheed [1] and M. H. D. Bassam Kurdy [2]

[1]*Department of Web Technologies, Syrian Virtual University, Damascus, Syria*
[2]*Department of Artificial Intelligence, Syrian Virtual University, Damascus, Syria*

Correspondence should be addressed to Aref Shaheed; aref_90718@svuonline.org

Received 22 May 2021; Revised 12 July 2021; Accepted 26 March 2022; Published 6 June 2022

Academic Editor: David Megias

Web application security has become a major requirement for any business, especially with the wide web attacks spreading despite the defensive measures and the continuous development of software frameworks and servers. In this study, we present a proposed model for a web application firewall that used machine learning and features engineering to detect common web attacks. Our proposed model analyses incoming requests to the webserver, parses these requests to extract four features that describe completely HTTP request parts (URL, payload, and headers), and classifies whether a request is normal or an anomaly. We took into consideration the limitation of previous works that use URL and payload only in classification and provided five features that describe and summarize all parts of the HTTP request using features engineering and previous experience in the field of the software security domain. Extracted features are length of request, percentage of characters allowed, percentage of special characters, and attack weight. These features were calculated for four different datasets CSIC 2010, HTTPParams 2015, Hybrid dataset (CSIC 2010 and HTTPParams), and real logs for the compromised web server. We evaluated our proposed model by using these updated datasets with four classification algorithms (Naive Bayes, logistic regression, decision tree, and support vector machine) with two methods (train test split and cross-validation) to negate the probability of overfitting and ensure that features are effective. Features values for a normal request are usually short request length, large allowed character ratio, small special character ratio, and zero attack weight or close to zero. Features values for anomaly requests are large request length, small allowed character percentage, large special character percentage, and very large numerically attack weight. Our proposed model achieved a classification accuracy of 99.6% with datasets used in research studies in this field and 98.8% with datasets of real web servers.

1. Introduction

Cyberattacks targeting web servers and applications were and still is one of the important points that are taken into consideration when an organization uses technology in its various types of work (applications, operating systems, databases, networks, etc.), and these attacks remain high risk despite the great diversity in the methods of combating them. This limited the impact of these attacks but was unable to make a tangible effect.

Despite the implementation of defensive measures by web application developers, attacks are constantly evolving, and there has become an urgent need for dedicated software or product that supports these defensive procedures and works in an integrated manner with these defensive procedures to raise the security level of web applications [1]. Security projects and standards were published to help developers and white hat hackers to increase the security level such as OWASP [2].

Traditional firewalls interact with packets in network and transport layers [3], while web application firewalls interact with web requests in the application layer [4]. These firewalls were operated using the signature [5], as they recognize the attack through a distinct fingerprint of it, and this requires large databases and storing the fingerprint of each attack after it is executed. Reliance on databases (signature-based protection) and hardcoded logic and rules (using traditional programming) make it more difficult to take advantage of expert knowledge by transferring it to the computer.

In recent decades, artificial intelligence has become a scientific revolution [6] and has achieved peerless superiority in mastering the work that humans do, and we think that a computer cannot learn and make decisions like humans, but rather it has become a competitor to human capabilities. In the coming decades, it is expected that artificial intelligence would eliminate many human jobs[7].

Researchers and information security professionals have specifically moved to harness the capabilities of artificial intelligence to detect and combat attacks [8]. The time has come for the machine to work side by side with the human to do what is difficult for him despite having hundreds of millions of real neurons.

Most recent works relied on one dataset only and work with URL and payload only. In this article, we used features engineering to present four generalizable features that summarize the whole HTTP request information (URL, payload, and headers) and we used four classification algorithms in machine learning in the classification phase to evaluate our proposed model.

The rest of this article is organized as follows: Section 2 presents materials and methods (related works and proposed model), Section 3 discusses results and discussion, Section 4 gives the conclusion, and Section 5 contains future work.

2. Materials and Methods

2.1. Related Works. Vulnerabilities of web applications have not changed as concepts; it changes in how to exploit them. The most popular vulnerabilities of web applications are as follows:

(i) Injections: manipulating the input to force a web application to execute arbitrary commands in the operating system and queries in databases [9], SQL injection is the most famous of injection attacks [10], and it allows the attacker to interact with the database by reading, writing, and modifying records.

(ii) Broken authentication: exploiting logical and weakness points in the authentication mechanism to takeover and control accounts [11].

(iii) Sensitive data exposure: manipulating a web application to make it throw exceptions and expose sensitive data such as credentials of the database [12].

(iv) XML external entity (XXE): manipulating inputs using functions that parse XML to execute arbitrary commands [13].

(v) Broken access control: accessing unauthorized resources in a web application due to the weakness of access control rules such as accessing the administrator panel if there is no restriction on access to it [14].

(vi) Security misconfigurations: using brute force to find and exploit security misconfigurations such as unpatched flaws, default configurations, unused pages, unprotected files and directories, and unnecessary services [15].

(vii) Cross-site scripting (XSS): injecting JavaScript code in a web application to modify the display of a web application and force the victim to execute it in his browser; there are many types of it, such as Reflected XSS and DOM XSS [16].

(viii) Insecure deserialization: manipulating the inputs of a web application by deserializing it, modifying it, and serializing it again to compromise the web application [17].

(ix) Using components with known vulnerabilities: stop updating the used component in a web application allows attackers to exploit its known vulnerabilities; this type of vulnerability is found in abundance, especially in CMS web applications [18].

(x) Insufficient logging and monitoring refer to the lack of logging and monitoring mechanisms and techniques, which allow attackers to find and exploit without being detected [19].

Studies and research about protecting web applications from malicious requests were following two methodologies to detect an attack: identify and detect a particular attack (such as detection of the SQL injection attack only or cross-site scripting attack detection) or classify requests if it is an anomaly or normal in general, regardless of the type of attack.

It also followed two approaches to transfer this experience to computers: designing and implementing behavioral-based detection by using artificial intelligence techniques such as classification algorithms or using a custom algorithm, and signature-based detection by using databases that contain patterns of attacks.

Most of the studies relied on old datasets such as CSIC 2010 [20], ECML-PKDD 2007 [21]. The proposed models were not evaluated using modern datasets, and the datasets created by some researchers are not available online.

Zhang et al. proposed a framework to detect web attacks by extracting seven features, web resource, attribute sequence, attribute value, HTTP version, header, and header input value. This framework includes three components: the probability distribution model, the hidden Markov model, and the one-class SVM model. Each of these components can be considered as a machine learning model. Each model trained on a dataset contains normal requests only and is evaluated by using two datasets: Wikipedia access traces [22] and FuzzDB [23]. Using a multimodel-based method takes advantage of all models in it, by this method, the authors mitigated false positive issue significantly [24]. The main advantage of this model is that it takes advantage of multiple components by combining them in a hybrid one. On the other side, running multiple components may affect performance. WAF is a real-time service that handles many requests and performance is an important parameter to take into consideration.

Tekerek and Bay provided a hybrid model for the detection of malicious and normal requests using signature-based detection to overcome speed issue for traditional attacks and behavioral-based detection to solve zero-day attacks issue using neural networks with three features described by mathematical equations as input to this neural network [25]. The advantages of this model are that it used hybrid detection methods (signature-based and behavioral-based) in detection that increase the performance and speed of implementation. In addition to the simplicity and speed of implementation of the neural network, this research has undergone successive development work in recent years and the model is mature. On the other side, the features extracted cannot be generalized for all web applications (values of the features calculated depends on average and derivation of other requests, it requires a massive log of an application to make the model able to classify the requests arrived at this application). In addition, the usage of statistical functions gives anomaly cases that cannot be handled (using the average in the denominator of 2 generates a very big number of requests that its length is near to the average of request lengths).

Sharma et al. used features engineering to extract seven features from the incoming request and used three classification algorithms to test their effectiveness. They applied preprocessing procedures on CSIC 2010 dataset to identify subcategories for malicious requests to overcome missing features issue [26]. This method yielded many useful features, but some of these features cannot be extracted from the dataset that used in this article due to the unavailability of its fields in the dataset, such as the length of cookies (you can review the structure of the CSIC 2010 dataset). In addition, researchers did not use another dataset and relied only on CSIC 2010.

Vartouni et al. applied n-gram character-based model to construct features. The size of features increased depends on the value of *n*; to overcome this issue and avoid using dimensionality decrease techniques, they applied an autoencoder to extract features as a data abstract. Deep learning algorithms were used to work more effectively with extracted features. This proposed model gave a generalizable model, but the classification accuracy was low in comparison with other models. In addition to use a single dataset, refer CSIC 2010 [27].

Hoang used supervised machine learning (inexpensive decision tree algorithm as a suitable real-time classifier to mitigate performance issue in terms of speed) to detect four major web attacks (SQLi, XSS, command injection, and path traversal) and N-gram used with fixed n value ($n = 3$) and PCA (principal component analysis) method to obtain a reduced number of features. He used the HTTPParams dataset [28] and CSIC 2010 to evaluate the proposed model and achieved high accuracy of 98.56% [29]. Hoang X.D proposed model takes web server logs as input and turns every single row into a vector, results of the model only focus on the HTTPParams dataset, and experiments done used only one algorithm.

Niu and Li extracted eight statistical features and used convolutional neural network (CNN) combined with gated recurrent unit (GRU) with CSIC 2010 dataset. Using these techniques together with eight features improved detection performance but at the expense of speed performance. They evaluated the detection performance of this model by comparing it with other deep learning methods and they achieved 99.00% accuracy [30]. Niu and Li also used only one dataset (CSIC 2010). In addition, using deep learning techniques for real-time services like WAF will affect speed performance if they deploy it in a real environment.

All of these previous studies deal with the request if it is normal or anomaly; it used the second methodology of detection.

Other studies detect specific attacks, such as detecting an attack of database rights; it used the first methodology, as the following study of Dr. Ahmad Ghafarian, in which he provided an approach that puts a line in each table and used an algorithm that it executes to verify queries before executing it on the web application. Malicious requests fetch the added line and the normal requests do not fetch it. This way, an SQL injection attack is detected in real time before it is executed on the web application, but the proposed model reveals only one of the seven types mentioned by the researcher in his article. In addition, this study did not discuss the overload on resources, databases, and the delay in execution time due to testing each query before executing [31].

There are many studies similar to the study of Dr. Ahmad Ghafarian that detect SQL injection in runtime like AMNESIA (proposed by Halfond and Orso) [32] and CANDID (proposed by Bisht, Madhusudan, and Venkatakrishnan) [33].

2.2. Proposed Model

2.2.1. Architecture. Our proposed model of WAF works as an operating system service, which acts as an intermediary between the web server and the clients. This service receives the request, parses it, extracts features, classifies it, and makes decisions based on the classification result.

WAF can be configured through a dedicated web application (web control panel, see Figure 1). The proposed WAF consists of the following five basic units (see Figure 2):

(1) Power on/off unit
(2) Training unit
(3) Parsing unit
(4) Classification unit
(5) Decision-making unit

The process starts in the first unit, the power on/off unit, when the WAF is running, the OS service contacts the database and fetches the configurations to run WAF, initiate the listener, and wait for incoming requests for the WAF that mediates between the client and the web server.

After running the WAF, the training process starts using the selected dataset and the selected classification algorithm.

After the completion of the training process and the completion of the work of the first and second units, WAF is ready to receive requests.

When the request arrives, WAF, the first unit to handle it is the parsing unit, which breaks down the request, extracts the features, and passes it as a vector to the classification unit that classifies according to the classification method that the administrator chooses it in the training unit.

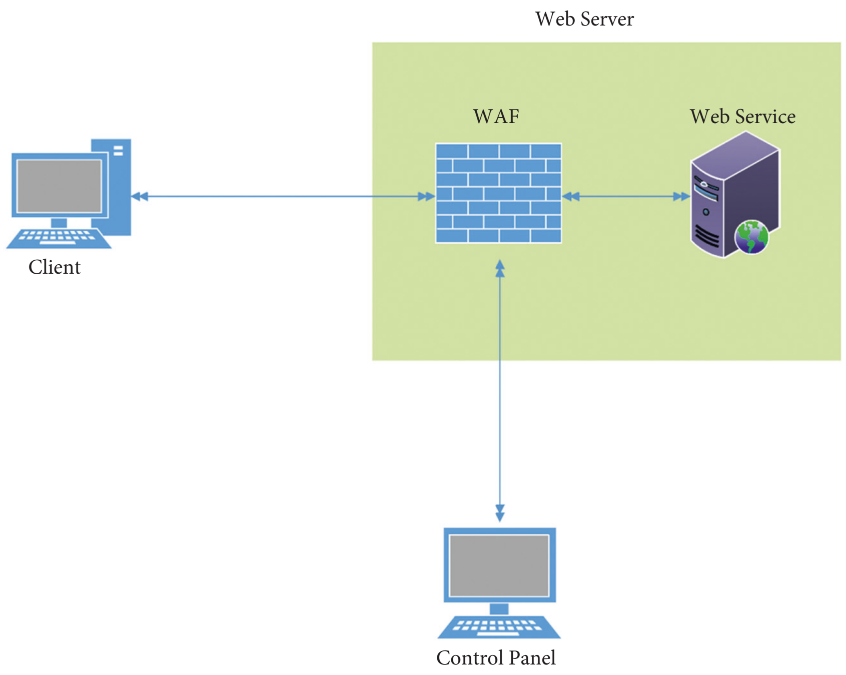

Figure 1: Architecture of the web server environment with proposal WAF.

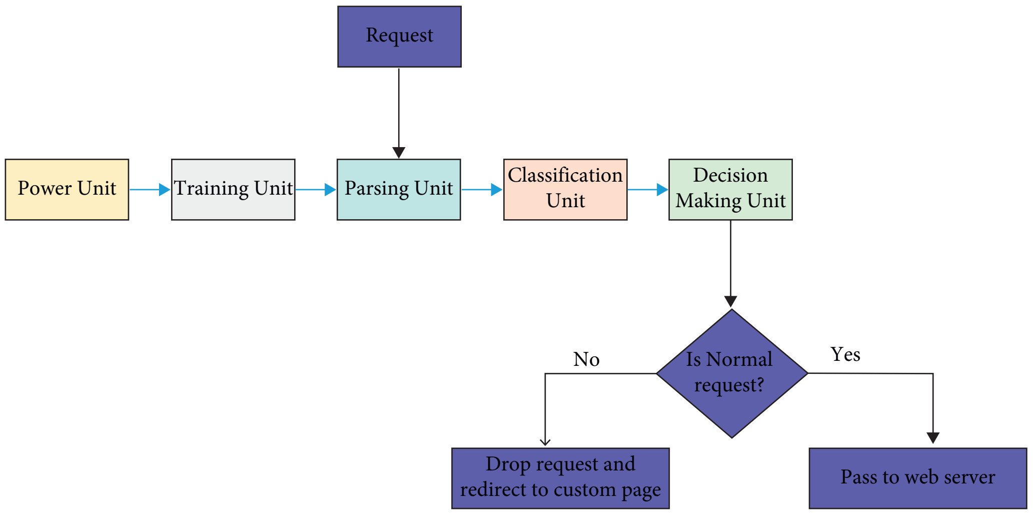

Figure 2: Units of proposal WAF (brief diagram).

After classifying the request, the classification unit sends the result to the decision-making unit, which takes the appropriate action to pass or drop the request (see Figure 3 or Algorithm 1).

(1) Power on/Off Unit. This unit is responsible for controlling WAF by turning it on and off. When the WAF started, all configurations will be fetched from the database such as the IP address and port of the firewall, IP address, and port of the web server to run the service (the listener).

(2) Training Unit. After running WAF, the dataset name and classification algorithm will be fetched from databases to train the model, now WAF is ready to receive requests.

We used popular classification algorithms, any algorithm can be added by inserting its name in the database (algorithms table) and call it programmatically in the WAF implementation code.

In addition, any dataset can be added by inserting its name in the database (datasets table) and adding the CSV file of the desired dataset in the dataset folder inside the WAF implementation code folder.

Used algorithms are Naive Bayes, Logistic Regression, Decision Tree, and Support Vector Machine (popular algorithms in binary classification problems) [34].

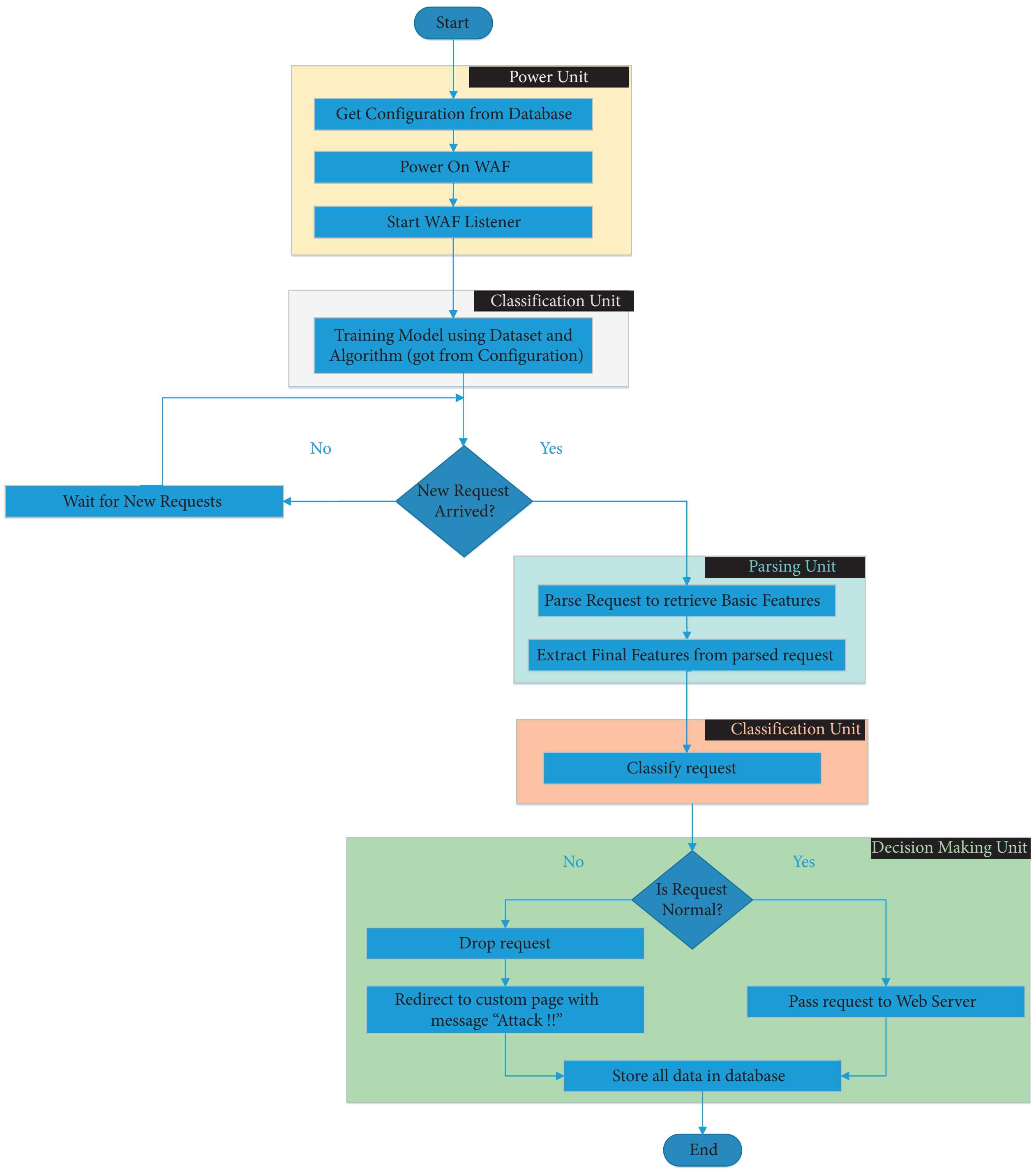
Start
Power Unit
Get Configuration from Database
Power On WAF
Start WAF Listener
Classification Unit
Training Model using Dataset and Algorithm (got from Configuration)
No
Yes
Wait for New Requests
New Request Arrived?
Parsing Unit
Parse Request to retrieve Basic Features
Extract Final Features from parsed request
Classification Unit
Classify request
Decision Making Unit
No
Yes
Is Request Normal?
Drop request
Redirect to custom page with message "Attack !!"
Pass request to Web Server
Store all data in database
End

Figure 3: Units of proposal WAF (detailed diagram).

We preferred to use these algorithms instead of neural networks, which can be used in future works (see Future works for Researchers section).

Used datasets include CSIC 2010, HTTPParams 2015, Hybrid dataset (Generated by combining CSIC 2010 and HTTPParams 2015), and Custom dataset (logs of real web servers).

(3) Parsing Unit. After turning on WAF and the training model, now WAF is ready to receive requests. When an HTTP request arrives at WAF, the parsing unit breaks down the request to extract features (features will be discussed in the next section).

The parsing unit creates a final vector consisting of features and passed this vector to the classification unit.

```
Input of data: $d$ (dataset), a (algorithm), p1 (web server port), i1 (web server IP), p2 (WAF port), i2 (WAF IP).
(1) Start
(2) Connect to database to initialize Inputs (d, a, p1, i1, p2, i2)
(3) Start WAF listener using Inputs (p1, i1, p2, i2)
(4) Training WAF using Inputs (d, a)
(5) While WAF listener is "ON":
(6)   If new request arrived R:
(7)     Parse R
(8)     Compute basic features vector B from parsed R
(9)     Compute V Final features vector from B
(10)    Compute C (class) of parsed request $R$ by classify based on V
(11)    If $C$ = "anomal"
(12)      Drop request
(13)      Redirect to custom page with message "Attack"
(14)    Else//$C$ = 'normal'
(15)      pass request to web server
(16)    Store V and C in database
(17)    Endif
(18)  Endif
(19) EndWhile
(20) End
```

Algorithm 1: Units of proposal WAF (detailed algorithm).

Parsing unit implemented using MITM Proxy in *Python* [35].

(4) Classification Unit. This unit receives the final vector from the parsing unit and classifies the request depending on it. The classification unit sends the classification results to decision-making unit.

(5) Decision-Making Unit. This unit receives the classification results from the classification unit and forwards the request to the web server if the request is normal, and drops the request if it is an anomaly.

2.2.2. Features Engineering. When an HTTP request arrives at the parsing unit, it is dismantled to extract the basic features, and these basic features will be used to calculate the final features that will be sent to the classification unit (see Algorithm 2).

(1) Basic Features. All basic information extracted directly from the request is called a basic feature; it is the content of HTTP Message [36], in our proposed model, and we have five basic features (see Table 1):

(1) HTTP Method: HTTP protocol, method, or verb that is used by the client to request the resource from the web server, it may be POST, GET, HEAD, OPTION, PUT, PATCH, or DELETE
(2) Absolute URL (URL): it includes the IP address or domain of a web application with the resource, for example, https://www.mysite.com/login
(3) Payload: all data submitted by the client (text input, dropdown menu, text area, etc.)
(4) Headers: request headers (original headers or custom headers added by the client or web application)
(5) Files: it is usually included in the payload but we separated it as an independent feature

(2) Final Features. Final features used by WAF to classify the request if it is normal or anomaly; these features are calculated and extracted based on the basic features, and we have four final features (see Table 2):

(1) Input length: it describes the number of characters in payload, and it is calculated as follows:

$$l = \sum_{i=0}^{n} c_i, \tag{1}$$

where l is the input length, c is the character in payload, and n is the payload length.

Usually, this feature value is bigger in anomaly requests compared to normal requests (see Figure 4).

(2) Alphanumeric character ratio: it describes the ratio of alphanumeric characters over the input length. Normal requests usually contain more numeric and alphabetic characters compared to special characters such as symbols, so this feature will have a big value in normal requests compared to anomaly requests (see Figure 5).

This feature is calculated as follows:

$$a = \sum_{i=0}^{n} \frac{(c_i | c_i \in e)}{l} \times 100, \tag{2}$$

```
    Input of data: $R$ raw HTTP request
    Output: $V$ Final features vector
(1) Start
(2) parse $R$
(3) compute $B$ basic features vector from parsed $R$
(4) compute $V$ Final features vector from $B$
(5) **End**
```

Algorithm 2: Features extraction from raw HTTP request by parsing unit.

Table 1: Basic features of the proposed model.

Feature	Description
HTTP method	HTTP protocol method
URL	IP address or domain of web application with resource
Payload	All data submitted from the client
Headers	Request headers
Files	Uploaded files in payloads

Table 2: Extracted features of the proposed model.

Feature	Description
Input length	Number of characters in payload
Alphanumeric character ratio	The ratio of alphanumeric characters over input length
Special characters ratio	The ratio of nonalphanumeric characters over input length
Attack weight	Sum of five subfeatures (see Table 3): (i) URL weight (ii) Number of attack words in inputs (iii) Manipulate payload weight (iv) Alphanumeric character to special character ratio (v) Files weight

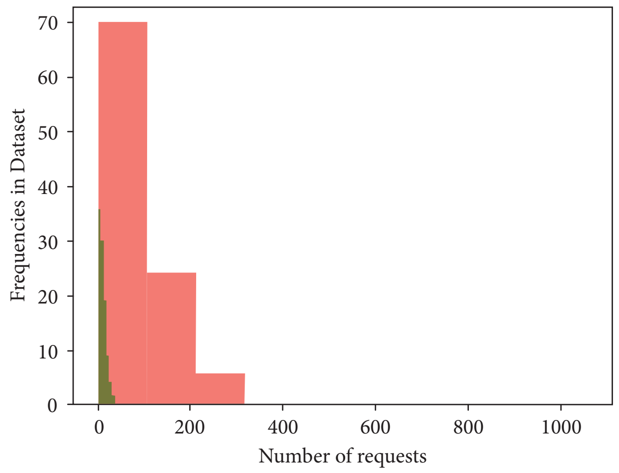

Figure 4: Histogram implementation of input length feature in the CSIC 2010 dataset; green values represent normal request and red values represent anomaly requests.

where a is the alphanumeric character ratio, c is the character in payload, n is the payload length, e is the cluster of allowed characters (alphabet and numbers), and l is the input length.

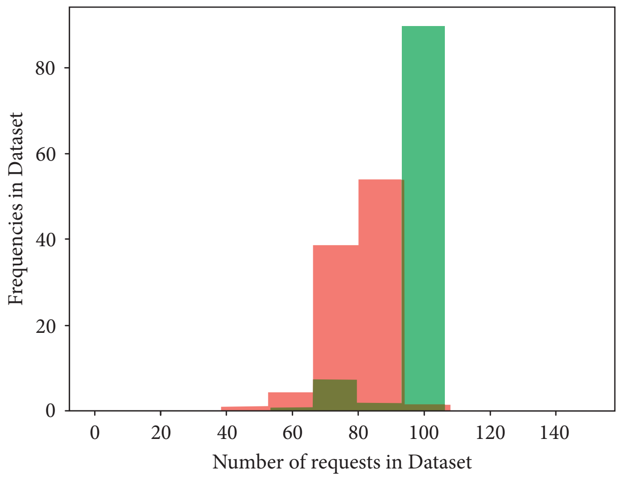

Figure 5: Histogram implementation of alphanumeric character ratio feature in the CSIC 2010 dataset; green values represent normal request and red values represent anomaly requests.

(3) Special character ratio: it describes the ratio of special characters (nonalphanumeric) over the input length.

Anomaly requests usually contain fewer numeric and alphabetic characters compared to special characters such as symbols, so this feature will have a big value in anomaly requests compared to normal requests (see Figure 6).

This feature is calculated as follows:

$$s = \sum_{i=0}^{n} \frac{(c_i | c_i \in f)}{l} \times 100, \quad (3)$$

where s is the special character ratio, c is the character in payload, n is the payload length, f is the cluster of not allowed characters (any character that is not alphabet and numbers), and l is the input length.

Alternatively, it can be calculated as follows:

$$s = 1 - a, \quad (4)$$

where s is the special character ratio and a is the alphanumeric character ratio.

(4) Attack weight: it is the most important feature in the classification process. It is calculated by summing four subfeatures.

Anomaly requests usually have a big attack weight compared to normal requests (see Figures 7 and 8).

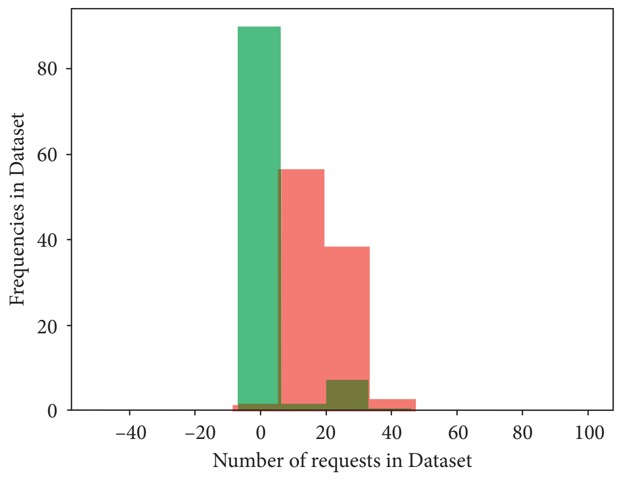

FIGURE 6: Histogram implementation of special character ratio feature in the CSIC 2010 dataset; green values represent normal request and red values represent anomaly requests.

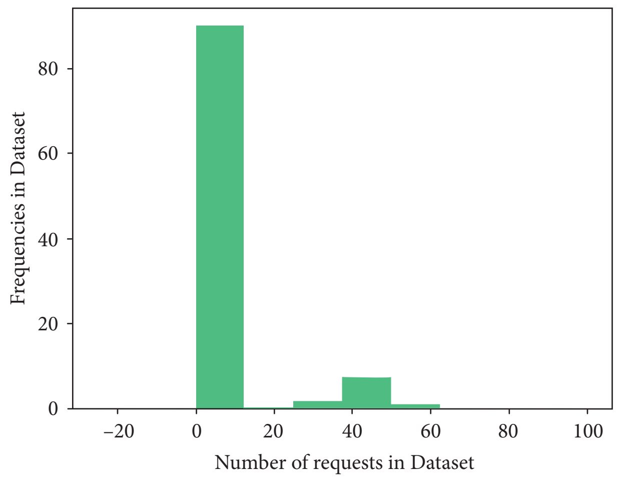

FIGURE 7: Histogram implementation of attack weight feature in the CSIC 2010 dataset for normal request and red values represent anomaly requests.

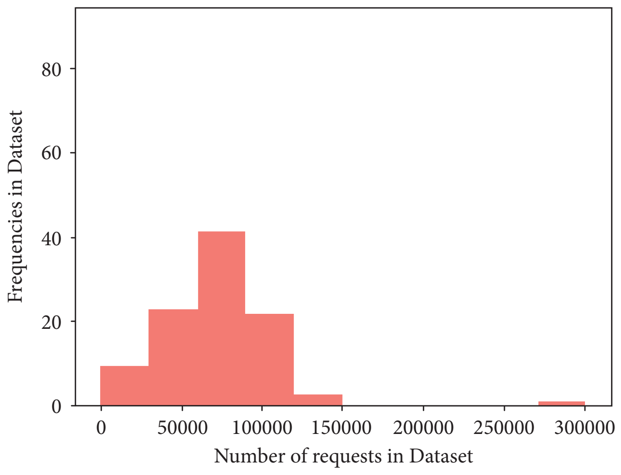

FIGURE 8: Histogram implementation of attack weight feature in CSIC 2010 dataset for anomaly request.

(1) URL weight
The value of this subfeature is initialized to zero. Value of this subfeature for each discovered special character, attack word, or access of unauthorized resource.
Each discovered malicious have its weight.
This subfeature is calculated as follows:

$$u = \sum_{i=0}^{n} (w_i | d_i \in URL), \tag{5}$$

where u is the URL Weight, d is the discovered malicious, w is the weight of discovered malicious, n is the number of discovered malicious in WAF database, and URL is request absolute URL.
Example: http://www.example.com/.env.
u = 200 (200 is the weight of discovered access of unauthorized resource—only one discovered malicious in URL).

(2) Number of attack words in inputs
The value of this subfeature is initialized to zero. Value of this subfeature for each discovered attack word in payload and headers.
Each discovered word attack has its own weight.
This subfeature is calculated as follows:

$$v = \sum_{i=0}^{n} (w_i | d_i \in \text{Input}), \tag{6}$$

where v is the number of attack words in inputs, d is the discovered attack word, w is the weight of attack word, n is the number of attack words in WAF database, and Input is request headers and payloads.
Example: Email = aref@mail.com & passwd=' or 1=1 --&mode=' or 1=1 --
v = 150 + 150 (150 is the weight of discovered SQLI, it exists twice).

(3) Manipulate payloads weight
The value of this subfeature is initialized to zero. The value of this subfeature increases for each discovered manipulate in payload and headers.
Manipulation is passing wrong data to the application to throw an exception and to expose sensitive data, for example, passing a string as a mobile number.
Each discovered manipulation has its own weight.
This subfeature is calculated as follows:

$$m = \sum_{i=0}^{n} (w_i | d_i \in \text{Input}), \tag{7}$$

where m is the manipulate payload weight, d is the discovered manipulation, w is the weight of discovered manipulation, n is the number of manipulations in WAF database, and Input is request headers and payloads.

Example: Email = aref@
mail.com&mobile = Hello
m = 100 (100 is the weight of discovered manipulation in payload, passing a string as a mobile number—only one discovered manipulation in URL).

(4) Alphanumeric characters to special character ratio
The value of this subfeature is the result of dividing alphanumeric character ratio over special characters ratio.
If alphanumeric character ratio or special character ratio equals zero then set subfeature value to zero.
This subfeature is calculated as follows:

$$r = \begin{cases} 500, & \left(\dfrac{s}{a}\right) \geq 0.3, \\ 0, & \left(\dfrac{s}{a}\right) < 0, \end{cases} \quad (8)$$

where r is the alphanumeric character to special character ratio, s is the special character ratio, and a is the alphanumeric character ratio.

Example: Email = aref@mail.com&passwd = ' or 1 = 1 -- &mode = ' or 1 = 1 --
a = 40/55 (all numbers and alphabet number over input length).
s = 15/55 (count of nonalphanumeric characters such as' = - @. and)
$s/a = 0.375$
r = 500.

(5) Files weight
The value of this subfeature is initialized to zero. Value of this subfeature increases for each suspicious discovered in files.
Suspicious cases:
(i) Invalid file extension (.exe,.bin,.php, etc.)
(ii) Positive results of antivirus scanning (we used three antiviruses: Kaspersky, MalwareBytes, and BitDefender)
Each discovered malicious have its weight.
This subfeature is calculated as follows:

$$f = w1 + w2 + w3 + w4, \quad (9)$$

where f is the files weight, w_1 = (300 if file extension is invalid or 0 if file extension is valid), w_2 = (200 if Kaspersky detect this file as virus or 0 if not), $w3$ = (200 if MalwareBytes detect this file as virus or 0 if not), and $w4$ = (200 if BitDefender detect this file as virus or 0 if not).
Example: uploaded file: shell.php
w1 = 300 (invalid extension).
w2 = 200 (Kaspersky detected it as virus).
w3 = 200 (MalwareBytes detected it as virus).
w4 = 200 (BitDefender detected it as virus).
f = 300 + 200 + 200 + 200 = 900.
You have to calculate it for each uploaded file and sum the results to get the final files weight for all files in request as follows:

$$F = \sum_{i=0}^{n} f_i, \quad (10)$$

where F is the files weight, n is the number of files in request, and f is the file weight.

Finally, the value of the attack weight feature is calculated by summing all subfeatures as follows:

$$z = u + v + m + r + F, \quad (11)$$

where z is the attack weight, u is the URL Weight, v is the number of attack words in inputs, m is the manipulation payload weight, r is the alphanumeric character to special character ratio, and F is the files weight.

Now, all requests are converted from the raw form to the final form (four features with the label).

This table is a sample of the dataset generated by our model, label is 1 for anomaly requests and 0 for normal requests (see Table 4):

3. Results and Discussion

A set of generalizable features extracted from HTTP requests to detect common attacks on web applications.

We used four datasets: CSIC 2010, HTTPParams 2015, Hybrid dataset (CSIC + HTTPParams), and custom web server logs (compromised real server). The last dataset was not published in the GitHub repository due to its privacy.

We used four basic extracted from HTTP requests to calculate the final features, basic features are HTTP Protocol (HTTP Method), Absolute URL (URL), payload, headers, and files. Extracted features are the length of request, percentage of characters allowed, percentage of special characters, and attack weight.

We used various classification algorithms that work more efficiently on binary classification problems, such as Linear Regression, Decision Tree, and Naive Bayes but we focus on Naive Bayes.

3.1. Experiments

3.1.1. Experimental Environment. WAF implemented in web server under Linux Xubuntu 20.04 LTS. This server contains apache2 service (web service), web control panel (web application developed using Django-*Python*), and WAF service (daemon service).

3.1.2. Preprocessing Datasets. Four datasets were used in this study: CSIC 2010, HTTPParams, Hybrid dataset (CSIC 2010 and HTTPParams), and custom dataset of compromised web server logs. Data preparation procedure has been done on these datasets (remove missing values, duplications, and outliers) and exported as CSV to be able to interact with machine learning algorithms in *Python* (Scikit-learn package).

Table 3: Subfeatures of attack weight feature.

Subfeature	Description
URL weight	Sum of weights of discovered manipulation in URL
Number of attack words in inputs	Sum of weights of discovered attack words in inputs
Manipulate payload weight	Sum of weights of discovered manipulation in payloads
Alphanumeric character to special character ratio	The ratio of the number of alphanumeric characters to the number of nonalphanumeric characters
Files weight	Sum of weights of the malicious files (malicious file weight is the weight of extension + sum of weights of scan using three antiviruses)

Table 4: Sample of the dataset used to train the model, we have four features for every single request and its label.

payload_len	Alpha	non_alpha	attack_feature	Label
0	0	0	0	**0**
41	95.45454545	4.545454545	200	**1**
241	100	0	0	**0**
9	100	0	0	**0**
24	94.73684211	5.263157895	2600	**1**
54	77.77777778	22.22222222	90000	**1**
75	100	0	0	**0**
103	87.37864078	12.62135922	60000	**1**
91	84.61538462	15.38461538	90000	**1**

CSIC 2010 contains 18 columns, custom compromised web server logs contain 10 columns, and HTTPParams 2015 contains 4 columns.

All previous datasets after preprocessing and dimension reduction procedures become with only 5 numeric columns, all columns that contain information about request were removed and replaced by final features columns that describe request briefly and effectively. Thereafter, the Hybrid dataset became very simple.

3.1.3. Training. We used four algorithms to classify (Naive Bayes, Logistic Regression, Decision Tree, and SVM). Four datasets were fed to the classifier using two methods: train test split (80%, 20%) and cross-validation (100 Folds), and results were very close.

Mixing and shuffling rows of CSIC 2010 and HTTPParams 2015 as a new dataset (Hybrid dataset) gave a very close result compared to the results of the classifier with each of the datasets separately (see Table 5).

Previous experiments negate the probability of overfitting and prove that the final features of our proposed model are effective.

3.2. Results

3.2.1. Results Based on Datasets. Our proposed model used Naive Bayes with cross-validation (100 Folds) and achieved an accuracy of 98.8% with the dataset created from logs of a compromised real web server, 97.61% with HTTPParams dataset, 99.58% with CSIC dataset, and 96.40% for Hybrid dataset (combination of CSIC 2010 and HTTPParams 2015).

Usage of four different datasets negates the probability of overfitting presence, to confirm that, k-fold cross-validation used in training also [37].

Most of the related works used CSIC 2010 dataset with or without the custom dataset, and we used it in the proposed model for the possibility of comparing the proposed model with previous models (see Figure 9 or Table 5).

Implementation of the proposed model includes a function to export WAF records as a new dataset with the ability to correct records. Administrators can train the proposed model using this exported dataset to strengthen WAF in protecting its web applications.

Most false positive cases are normal requests classified as anomaly requests (not the opposite).

3.2.2. Results Compared to Related Works. Our proposed model achieved high accuracy of 98.8% compared with related works. The following table shows the results for CSIC 2010, HTTPParams, and custom datasets created by the researchers (see Figure 10 or Table 6).

3.3. Comparison

3.3.1. Limitations of Previous Works. Researchers have provided many models for detecting web attacks, and despite their various features, there are some common weaknesses among these researches, which can be summarized as follows:

(1) Extracted features are not able to be general features and most of these features fit only web applications, which it extracts from it.

(2) Using old datasets such as CSIC and evaluating the model depends on the results of training it. In addition, all modern datasets used are not available on the Internet.

TABLE 5: Classification accuracy of our proposed model for various datasets using Naive Bayes.

	CSIC 2010	HTTPParams 2015	Hybrid dataset	Compromised web server dataset
Number of normal requests	28,800	19,305	48,105	60250
Number of anomaly	11,213	11,764	22,977	5210
Classification accuracy (80% training, 20% testing)	99.59%	97.91%	96.40%	98.80%
Classification accuracy (100-fold cross-validation)	99.71%	98.02%	96.66%	98.97%
False positive rate	0.54%	1.20%	3.35%	0.84%

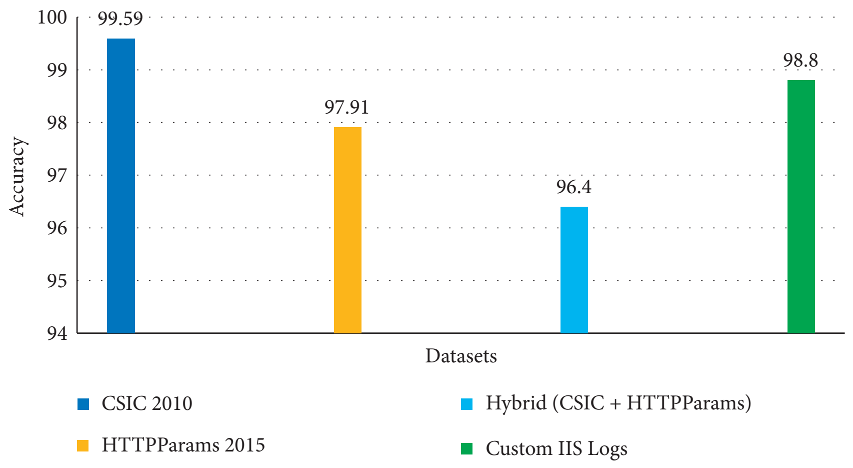

FIGURE 9: Classification accuracy of our proposed model for various datasets.

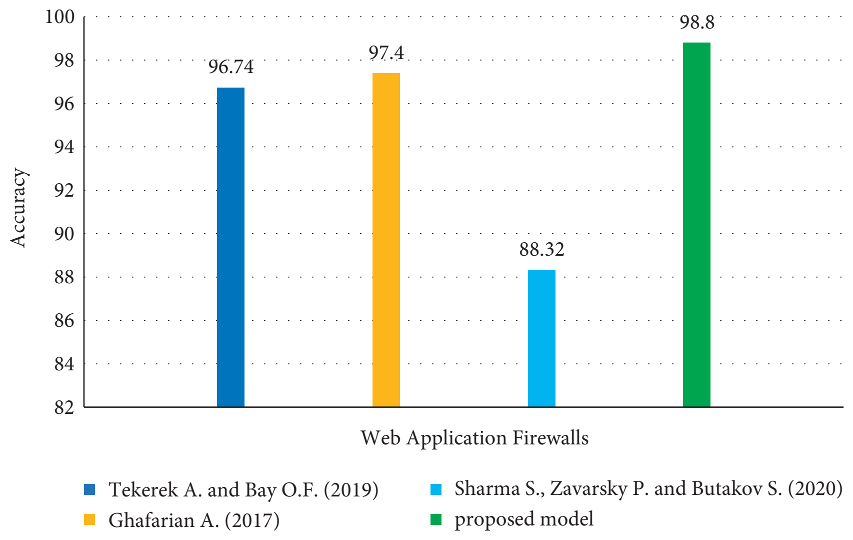

FIGURE 10: Classification accuracy of our proposed model compared with related works.

TABLE 6: Classification accuracy of our proposed model compared with related works.

	Our proposed model	Tekerek and Bay [25]	Sharma et al. [26]	Ghafarian [31]
CSIC 2010	99.59%	96.74%	94.7%	88.32%
ECML-PKDD 2007	Not tested	94.53%	Not tested	Not tested
HTTPParams 2015	97.61%	Not tested	Not tested	Not tested
Custom dataset	98.8%	98.52%	Not tested	Not tested

(3) Some papers of related works contain some errors and inaccurate information, such as the study by Sharma S., Zavarsky P., and Butakov S. (2020) [26]. They used features that cannot be extracted from CSIC 2010 (e.g., _cookie_len feature).

(4) Most of the related works process payload only without taking headers and files into consideration.

(5) Hybrid models are too rare (in related works only Tekerek A. and O.F. Bay (2019) paper is a hybrid model).

(6) Most of the related works detect common web attacks such as XSS and SQLI, no suggested model can detect attacks that use normal requests to be performed, such as DOS attacks.

3.3.2. Advantages of the Proposed Model. Disadvantages of related works and weakness points were taken into consideration while designing and preparing our proposal model. Features extracted in this model are general and can work with any web application. In addition, we used various datasets (standard datasets such as CSIC 2010 to compare our model with related works, modern datasets such as HTTPParams 2015, and Hybrid dataset, in addition, we also used a custom dataset of a real compromised web server). Final features describe all parts of the HTTP request including headers and files. Finally, a high-accuracy rate was achieved (98.8% for custom dataset and 99.6% for standard dataset).

4. Conclusion

In this article, we proposed a web application firewall model that used machine learning techniques and features engineering to detect common web attacks. We took into consideration major limitations in previous works (unuse of request headers, using one dataset only, absence of general features). Features engineering and previous experience in the software security domain were used to extract general and comprehensive features that describe and summarize requests and make the classification problem much easier. We extract the final four features from HTTP requests using basic features. **Basic features:** All basic information extracted directly from the request, we have five basic features: HTTP protocol (HTTP method), absolute URL (URL), payload, headers, and files. Final features: all features that are calculated and extracted based on basic features, we have four extracted features: input length, alphanumeric character ratio, special character ratio, and attack weight. Values of extracted features for the normal request are usually short request length, big allowed character ratio, small special character ratio, and zero risk weight or close to zero. Values of extracted features for anomaly requests are usually large request length, small allowed character percentage, large special character percentage, and very large numeric risk weight. To increase the security level, we suggest training our proposed model on web server records of web applications that will be protected by WAF. Any classification algorithm can be used, but we used algorithms that work more efficiently on binary classification problems, such as Logistic Regression, Decision Tree, and Naive Bayes, we focus on Naive Bayes. Our proposed model achieved a high classification accuracy of 99.6% with standard datasets used in research studies in this field (CSIC 2010), and 98.8% with datasets of real compromised web server dataset.

5. Future Works

Future works domains are wide, it can be summarized as follows (see next three subsections for more information):

Researchers in the information security domain can develop proposed WAF by feeding it with more datasets (generated dataset from our proposed WAF or by creating a custom dataset from web servers logs) or by add or modify current features, also they can develop separate components and migrate them with our proposed WAF (signature-based model to check request before pass it to the classifier, DOS attack detector, and use natural language process to make a model to identify attack words instead of using a table in the database to store these attack words).

For software engineers, developers and information security engineers use our proposed model to evaluate their applications and improve their skills by learning how to write a secure source code.

Sponsors and businesspersons can invest money to develop the proposed model and become a commercial product.

5.1. Future Works for Researchers in the Information Security Domain

(1) Export web server logs as dataset after deploying WAF for a specified period in the real environment, and use neural networks instead of algorithms used in this article.

(2) Use natural language processing to generate rules to detect common attack words and malicious payloads instead of using hardcoded arrays. Common attack words and malicious payloads are implemented as arrays within the proposed model.

(3) Use reinforcement learning to obtain feedback and use it during decision-making (this proposal can be implemented after the proposed model becomes mature and ready, so reinforcement learning is not a good option in real-time applications).

(4) Use various and new datasets (dimensionality reduction required depends on features numbers [38, 39]).

(5) Extend the proposed model to detect attacks that use normal requests such as DOS attacks and brute force attacks. The proposed model cannot detect these types of attacks because it can detect attacks by detecting anomaly requests.

(6) Use features engineering to modify or add features to the model. Features are the most important component in the model depending on the researcher's experience in the field of information security.

(7) Extend the proposed model by adding components that work with signature-based detection. The proposed model works by using one technique, which is detection depending on the content through features extracted from the incoming request. The proposed system can be combined with different detection techniques and provide a hybrid model (see Tekerek A. and Bay O.F (2019) [25] study in related works).

(8) Use ensemble classifiers instead of the proposed classifier (Naive Bayes) to increase the efficiency of request classification [40] (it depends on balancing between security level sensitivity and performance—usually using ensemble classifiers increase the efficiency of classification at the expense of speed performance especially that WAF is a real-time service).

5.2. Future Works for Software Engineers, Developers, and Information Security Engineers

(1) Training model by supplying logs of their web applications as a dataset. This will increase the security level of WAF to protect their web application and WAF will get a great experience.

(2) Implementing the proposed model to support Windows operating systems (current implementation supports Linux distributions only).

(3) Installing vulnerable web applications such as DVWA in the web server and try to bypass WAF; this will increase the experience of information security engineers to learn new methods of bypassing WAF and help researchers to modify features to prevent these bypasses.

5.3. Future Works for Sponsors and Businesspersons. Investing money to develop the proposed model to be a product in the security and IT market.

Data Availability

CSIC 2010, HTTPParams 2015, and a hybrid dataset with *Python* code to train these datasets are available in the following repository: https://github.com/aref2008/waf. We recommend reading README.md to read all instructions about usage. We did not publish a custom dataset (real compromised web server logs) due to privacy.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

The authors thank everyone who provided any kind of support to complete this work, especially to the Hindawi Foundation, Mrs. Nadia Ali, the reviewers, and the editors Prof. Megias David, Prof. Roberto Di Pietro, and Prof. Jhaveri Rutvij for their great efforts.

References

[1] M. Choraś and R. Kozik, "Machine learning techniques applied to detect cyber attacks on web applications," *Logic Journal of IGPL*, vol. 23, no. 1, pp. 45–56, 2015.

[2] D. Wichers and J. Williams, "Owasp Top Ten," *The open web application security project*, vol. 3, 2017.

[3] Z. J. Huang, O. Ai, and X. U. Hong-xian, "Network Security and Firewall Technology," *Journal of Naval University of Engineering*, vol. 1, 2002.

[4] A. H. Yaacob, M. Nazrul, N. Ahmad, and M. Roslee, "Moving towards positive security model for web application firewall," *International Journal of Computer and Information Engineering*, vol. 6, no. 12, pp. 1763–1768, 2012.

[5] P. P. Mukkamala and S. Rajendran, "A survey on the different firewall technologies," *International Journal of Engineering Applied Sciences and Technology*, vol. 5, no. 1, pp. 363–365, 2020.

[6] W. Wang and K. Siau, "Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity," *Journal of Database Management*, vol. 30, pp. 61–79, 2019.

[7] M.-H. Huang and R. T. Rust, "Artificial intelligence in service," *Journal of Service Research*, vol. 21, no. 2, pp. 155–172, 2018.

[8] J. H. Li, "Cyber security meets artificial intelligence: a survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 12, pp. 1462–1474, 2018.

[9] A. K. Dalai and S. Kumar Jena, "Neutralizing SQL Injection Attack Using Server Side Code Modification in Web applications," *Security and Communication Networks*, vol. 2017, Article ID 3825373, 2017.

[10] D. Mitropoulos, V. Karakoidas, P. Louridas, and D. Spinellis, "Countering Code Injection Attacks: A Unified Approach," *Information Management & Computer Security*, vol. 19, no. 3, 2011.

[11] M. M. Hassan, S. S. Nipa, M. Akter et al., "Broken authentication and session management vulnerability: a case study of web application," *International Journal of Simulation: Systems, Science & Technology*, vol. 19, no. 2, pp. 1–6, 2018.

[12] J. Doshi and T. Bhushan, "Sensitive data exposure prevention using dynamic database security policy," *International Journal of Computer Application*, vol. 106, no. 15, pp. 18600–19869, 2014.

[13] A. A. Osincev and O. R. Laponina, "Vulnerability testing in web applications external entities XML," *International Journal of Open Information Technologies*, vol. 7, no. 10, pp. 71–79, 2019.

[14] D. H. Lee, J. W. Lee, and J. G. Kim, "Verification methods of OWASP TOP 10 security vulnerability under multi-tenancy web site's environments," *Convergence security journal*, vol. 16, no. 4, pp. 43–51, 2016.

[15] P. Jayabalan, R. Ibrahim, and A. A. Manaf, "Understanding cybercrime in Malaysia: an overview," *Sains Humanika*, vol. 2, no. 2, 2014.

[16] B. K. Ayeni, J. B. Sahalu, and K. R. Adeyanju, "Detecting cross-site scripting in web applications using fuzzy inference system," *Journal of Computer Networks and Communications*, vol. 2018, pp. 1–10, 2018.

[17] V. Pedreira, D. Barros, and P. Pinto, "A review of attacks, vulnerabilities, and defenses in industry 4.0 with new challenges on data sovereignty ahead," *Sensors*, vol. 21, p. 5189, 2021.

[18] N. Mendes, J. Duraes, and H. Madeira, "Benchmarking the security of web serving systems based on known vulnerabilities," in *Proceedings of the 2011 5th Latin-American Symposium on Dependable Computing*, pp. 55–64, IEEE, Sao Jose dos Campos, Brazil, April 2011.

[19] M. Malviya, A. Jain, and N. Gupta, "Improving security by predicting anomaly user through web mining: a review," *International Journal of Advances in Engineering & Technology*, vol. 1, no. 2, p. 28, 2011.

[20] C. T. Giménez, A. P. Villegas, and G. Á. Marañón, "Http dataset CSIC 2010," 2010, https://www.isi.csic.es/dataset/.

[21] F. Eisterlehner, A. Hotho, and R. Jäschke, "ECML/PKDD dataset," 2007, https://gitlab.fing.edu.uy/gsi/web-application-attacks-datasets/-/tree/master/ecml_pkdd.

[22] G. P. Urdaneta and G. V. S. Maarten, "Wikipedia access traces Datasets," 2008, http://www.wikibench.eu/?page_id=60.

[23] FuzzDB, 2007, https://code.google.com/p/fuzzdb/.

[24] M. Zhang, S. Lu, and B. Xu, "An anomaly detection method based on multi-models to detect web attacks," in *Proceedings of the 2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, pp. 404–409, IEEE, Hangzhou, China, December 2017.

[25] A. Tekerek and O. F. Bay, "Design and implementation of an artificial intelligence-based web application firewall model," *Neural Network World*, vol. 29, no. 4, pp. 189–206, 2019.

[26] S. Sharma, P. Zavarsky, and S. Butakov, "Machine learning based intrusion detection system for web-based attacks," in *Proceedings of the 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pp. 227–230, IEEE, Baltimore, MD, USA, May 2020.

[27] A. M. Vartouni, S. S. Kashi, and M. Teshnehlab, "An anomaly detection method to detect web attacks using stacked autoencoder," in *Proceedings of the 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, pp. 131–134, IEEE, Kerman, Iran, March 2018.

[28] "HttpParams dataset," 2015, https://github.com/Morzeux/HttpParamsDataset.

[29] X. D. Hoang, "Detecting common web attacks based on machine learning using web log," in *Proceedings of the International Conference on Engineering Research and Applications*, pp. 311–318, Springer, Thai Nguyen, December 2020.

[30] Q. Niu and X. Li, "A high-performance web attack detection method based on CNN-GRU model," in *Proceedings of the 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 804–808, IEEE, Chongqing, China, June 2020.

[31] A. Ghafarian, "A hybrid method for detection and prevention of SQL injection attacks," in *Proceedings of the 2017 Computing Conference*, pp. 833–838, IEEE, London, UK, July 2017.

[32] W. G. J. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," in *Proceedings of the 28th International Conference on Software Engineering*, pp. 795–798, Shanghai, China, May 2006.

[33] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "Candid," *ACM Transactions on Information and System Security*, vol. 13, pp. 1–39, 2010.

[34] R. Kumari and S. K. Srivastava, "Machine learning: a review on binary classification," *International Journal of Computer Application*, vol. 160, p. 7, 2017.

[35] M. Proxy: https://docs.mitmproxy.org/stable/.

[36] S. Suroto, "A review of defense against slow HTTP attack," *JOIV International Journal on Informatics Visualization*, vol. 1, no. 4, pp. 127–134, 2017.

[37] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: a review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[38] R. Abdulhammed, H. Musafer, A. Alessa, M. Faezipour, and A. Abuzneid, "Features dimensionality reduction approaches for machine learning based network intrusion detection," *Electronics*, vol. 8, no. 3, p. 322, 2019.

[39] G. T. Reddy, M. P. K. Reddy, K. Lakshmanna et al., "Analysis of dimensionality reduction techniques on big data," *IEEE Access*, vol. 8, pp. 54776–54788, 2020.

[40] G. T. Reddy, S. Bhattacharya, S. S. Ramakrishnan et al., "An ensemble based machine learning model for diabetic retinopathy classification," in *Proceedings of the 2020 International Conference on Emerging Trends in Information Technology and Engineering (Ic-ETITE)*, pp. 1–6, IEEE, Vellore, India, Feb 2020.