2022 First International Conference on Artificial Intelligence Trends and Pattern Recognition (ICAITPR) | 978-1-6654-2521-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICAITPR51569.2022.9844219

Web application reconnaissance scan detection using LSTM network based deep learning

Bronjon Gogoi
National Informatics Centre
Guwahati, Assam, India
asm-bronjon@nic.in

Rahul Deka
National Informatics Centre
Guwahati, Assam, India
r.deka@nic.in

Suchitra Pyarelal
National Informatics Centre
Guwahati, Assam, India
suchitra@nic.in

Abstract—**Web applications are frequent targets of attack due to their widespread use and round the clock availability. Malicious users can exploit vulnerabilities in web applications to steal sensitive information, modify and destroy data as well as deface web applications. The process of exploiting web applications is a multi-step process and the first step in an attack is reconnaissance, in which the attacker tries to gather information about the target web application. In this step, the attacker uses highly efficient automated scanning tools to scan web applications. Following reconnaissance, the attacker proceeds to vulnerability scanning and subsequently attempts to exploit the vulnerabilities discovered to compromise the web application. Detection of reconnaissance scans by malicious users can be combined with other traditional intrusion detection and prevention systems to improve the security of web applications. In this paper, a method for detecting reconnaissance scans through analysis of web server access logs is proposed. The proposed approach uses an LSTM network based deep learning approach for detecting reconnaissance scans. Experiments conducted show that the proposed approach achieves a mean precision, recall and f1-score of 0.99 over three data sets and precision, recall and f1-score of 0.97, 0.96 and 0.96 over the combined dataset.**

Keywords— Log Analysis, Cyber Security, Deep Learning, Application Security

I. INTRODUCTION

Web applications are becoming ubiquitous due to the numerous advantages that they have over other means of delivering services. Despite the advantages, web applications have are prone to different kinds of cyber-attacks from malicious threat agents. The malicious threat agents exploit vulnerabilities in web applications to steal sensitive information, modify and destroy data, hamper the availability of web applications to legitimate users through DoS and DDoS attacks, as well as cause defacement. Many of the malicious threat agents use automated tools to gather information, scan for vulnerabilities, and subsequently exploit web applications. Information gathering or reconnaissance is usually the first step undertaken by an attacker. The reconnaissance step will ultimately lead to the exploitation of the web application. Detecting reconnaissance scans earlier can enable web application administrators to take preventive and remediation measures before the attacker can proceed to the exploitation phase.

Traditional defences against web application attacks include WAF(Web Application Firewall), IDS(Intrusion Detection System) and IPS(Intrusion Prevention System). Web application firewalls block malicious traffic depending on either a positive or a negative security model. Intrusion Detection Systems looks for known attack patterns in the HTTP requests to identify attacks. Such approaches cannot detect rreconnaissance scans because the reconnaissance attacks do not include attack payloads or other features that can be detected using WAF or IDS. Traditional log analysis methods commonly use RPS (request per second) metrics to detect requests coming from an automated and high performance tool. Using RPS metric results in false positives as different websites will have different user characteristics depending on the type of the website. A social networking site that has a lot of external resources like css, js and images files and in which users are highly active will have a different average RPS than a banking site. Using RPS as a sole measure of detecting reconnaissance scans is not reliable. SIEM (Security Incident and Event Management) systems also use rules and signatures and hence can only detect known security events and cannot reliably detect reconnaissance attacks.

In this paper, an LSTM (Long Short Term Memory) network-based deep learning approach to analyzing web server access logs for the detection of reconnaissance scans is presented. The proposed approach models the behaviour of legitimate users as well as attackers using an LSTM network. All requests that do not conform to the behaviour of legitimate users are detected as reconnaissance scans and are reported, following which the web application administrators can take preventive or remediation measures to mitigate further exploitation of the web application. The contribution of this paper can be summarized as follows:

- An algorithm for modelling the behaviour of legitimate users in terms of HTTP requests by processing web server access logs.
- An algorithm for modelling the behaviour of attackers in terms of HTTP requests by processing web server access logs by attacking a web server with common web intrusion tools.
- Developing an LSTM based deep learning model to classify HTTP requests in web server access log as either attack or benign.

The rest of the paper is organized into sections. Section II presents the background, Section III presents related work, Section IV presents the proposed approach and, Section V

presents evaluation and finally, Section VI presents the conclusions.

II. BACKGROUND

This section provides background information on reconnaissance scans, HTTP access logs and existing approaches to detecting reconnaissance scans.

A. HTTP access logs

An HTTP access log is a file maintained by web servers to log HTTP requests coming from HTTP clients which can be browsers or any other application that supports the HTTP protocol. The access log file is organized into various fields and the specific format varies from web server to web server and the fields logged in the log file are configurable by the server administrator. Though the file format may differ, at the minimum the access file log contains the IP address of the HTTP client, the date and time of the request, the request method and the file requested from the web server. To perform analysis, this paper considers only the Apache web server which is the most widely used open source web server. The format of the Apache access log file is configurable but the most common formats include common log format and combined log format[1]. The log format specifiers used in the common and combined log formats are as follows –

- %>s – The HTTP status code for the request.
- %h – The remote hostname or the IP address.
- %l – The remote log name.
- %u – Remote user if the request was authenticated.
- %r – First line of the request.
- %b – Size of response in bytes.
- %{Referer} – Referring web site/web application.
- %{User-agent} – User-agent of the user.

The log format strings for the common and the combined log formats are shown in Table 1. In the proposed approach, access logs in the common log and the combined log formats are considered.

B. Reconnaissance Scans

Attackers use automated tools to perform reconnaissance scans on web applications. Reconnaissance consists of techniques that involve adversaries actively or passively gathering information that can be used to support targeting[2]. Reconnaissance scans analyze web applications and attempt to discover vulnerabilities that can later be exploited. Traditional web application defence mechanisms cannot easily detect and prevent reconnaissance scans, since reconnaissance scans don't include exploit payloads that can be matched with signatures by WAF's, IPS or IDS. Moreover, attackers can tune their tools to spoof the USER-AGENT string, limit the number of requests per second, etc. to make the traffic look like it is coming from a legitimate user. This makes it difficult for WAF's, IPS and IDS software to detect reconnaissance scans. Reconnaissance scans can be of many types, but in the proposed approach only

1. https://httpd.apache.org/docs/2.4/logs.html
2. https://attack.mitre.org/tactics/TA0043/

Table 1: Apache web server access log formats

Log Format	Format String
Common	"%h %l %u %t \"%r\" %>s %b" common
Combined	"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined

reconnaissance scans targeted at web applications are considered. A web application reconnaissance scan can be either an active scan or a vulnerability scan. In an active scan, the attacker probes the web application using a crawling tool and tries to gather as much information as possible about the web application. In a vulnerability scan, the attacker scans for vulnerabilities that can be exploited to gain access to the web. The proposed approach attempts to detect both active and vulnerability reconnaissance scans.

III. RELATED WORK

Web server access log analysis for attack detection has been widely studied by different researchers. In [1] T. Threepak and A. Watcharapupong used an entropy based analysis method for detecting web application attacks but not reconnaissance scan detection. M. Zolotukhin, T. Hämäläinen, T. Kokkonen, and J. Siltanen [2] proposed an unsupervised approach of detecting anomaly using SVDD, K-means, DBSCAN, SOM and LOF. A. Juvonen, T. Sipola, and T. Hämäläinen [3] proposed a machine learning based web application attack detection approach using random projection, pca(principal component analysis) and diffusion maps. M. Moh, S. Pininti, S. Doddapaneni, and T. S. Moh [4] used machine learning based on naïve bayes, and bayes net classifiers to detect SQL injection attacks.In [5] Q. Cao, Y. Qiao, and Z. Lyu used Decision Trees and Hidden Markov Model to detect web injection attacks. In [6] M. Baş Seyyar, F. Ö. Çatak, and E. Gül proposed a rule based attack detection system based on number of 404 status code, presence of SQL or XSS payloads in the URL.

Z. Zhang and C. N. Manikopoulos in [7] proposed a method of detecting UDP, TCP and ICMP reconnaissance scan by extracting statistical features and using a neural network for classification. H. U. Baig and F. Kamran proposed a method based on Time Independent Feature set for detecting network layer reconnaissance scans in [8]. In [9] H. U. Baig and F. Kamran used sequential networks for detecting port and network scan. In [10] J. Wang, M. Zhang, X. Yang, K. Long, and J. X used a density based clustering approach to detect web attacks.

The approaches mentioned in [1]-[10] can be classified in to two groups – the group that detect web application attacks using log analysis and the group that detects reconnaissance attacks in the network layer. In literature, no approach was found that attempts to detect reconnaissance scans against web applications.

IV. PROPOSED METHOD

The high-level architecture of the proposed method is shown in Figure 1. The proposed method is divided into two separate phases - data collection and training. The process of data collection, feature extraction. and training is described in sections IV(A) and IV(B), and IV(C) respectively.

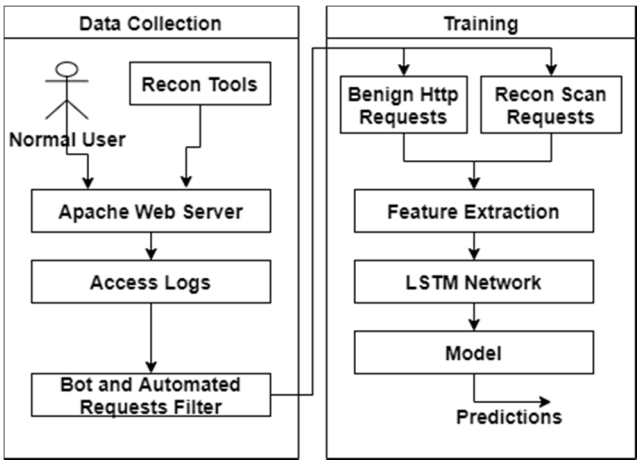

Figure 1: High level architecture of proposed system

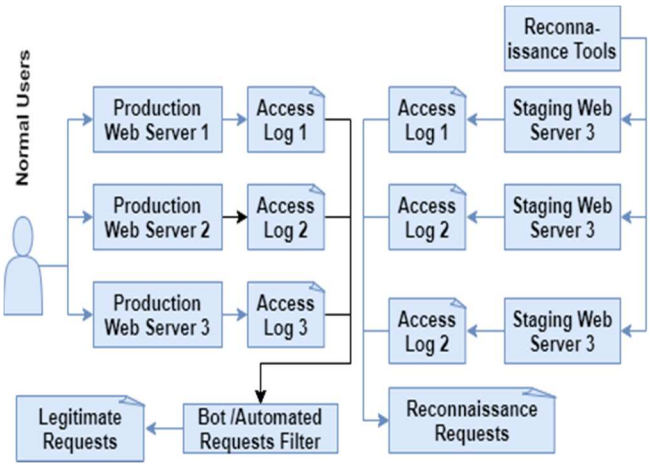

Figure 2: Data Collection Process

Table 2: Data set description

Dataset	Dataset Name	Class	Number of log records
Train Set	TS 1	Positive	4800000
		Negative	4500000
Test Set	DS 1	Positive	140000
		Negative	135000
	DS 2	Positive	120000
		Negative	115000
	DS 3	Positive	130000
		Negative	124500

A. Data Collection

During the data collection process, access logs are collected from a real production web server. The data collection process is shown in Figure 2. The access logs are then pre-processed to remove all HTTP requests which are generated by automated tools like bots and other crawlers. Removal of automated requests is necessary so that subsequent processes can closely model the behaviour of legitimate users. Algorithm 1 is used to filter automated requests by a bot and other automated tools. Algorithm 1 can eliminate more than 95% of automated requests.

To collect reconnaissance scans, a staging server is attacked using automated reconnaissance scan tools. The tools used for reconnaissance scans are Nikto[3], Wapiti[4], Whatweb[5], Owasp ZAP[6], BurpSuite[7]. The production web server is not used for reconnaissance scans so that legitimate users are not impacted by the reconnaissance scans. The access logs from both the production web server and the staging web server are separated into legitimate requests and reconnaissance scan requests. The legitimate requests form the negative class and the reconnaissance scans form the positive class.

3. Nikto: https://cirt.net/Nikto
4. Wapiti: https://github.com/wapiti-scanner/wapiti
5. Whatweb: https://github.com/urbanadventurer/WhatWeb
6. ZAP: https://owasp.org/www-project-zap/
7. https://portswigger.net/burp

Algorithm 1: Filter automated and bot requests from HTTP access log file

```
input ← HTTP access log file
output ← Filtered HTTP access log file
lines = split(input, "\n" )
ip_addresses = unique_ip(lines)
avg_rps = calculate_avg_rps()
for each ip in ip_address:
    requests_per_second[ip] = array()
end for
for line in input
    if line contains "bot" or "BOT"
        delete line from lines
end for
for each ip in ip_addresses:
    requests_per_second[ip] = calculateRps()
end for
for each requests_per_second:
    if requests_per_second[ip] > avg_rps:
        for each line in lines:
            delete line from lines
        endfor
end for
output ← lines
```

In the data collection phase, the number of negative samples is significantly higher than the number of positive samples. This is since reconnaissance events are rare events. As a result, the data set collected was imbalanced with more negative samples than positive samples. To solve this problem of an unbalanced data set, SMOTE (Synthetic Minority Oversampling Technique) [11] was used to generate synthetic samples for the minority class, which is in this case the positive samples. The data set after the application of SMOTE is then divided into train and test sets in the ratio 70:30, where 70 per cent will be used for training and 30 per cent for testing. There is no overlap between the train set and the test set. The test dataset is further divided into three sets DS 1, DS2 and DS3. The description of the dataset is given in Table 2.

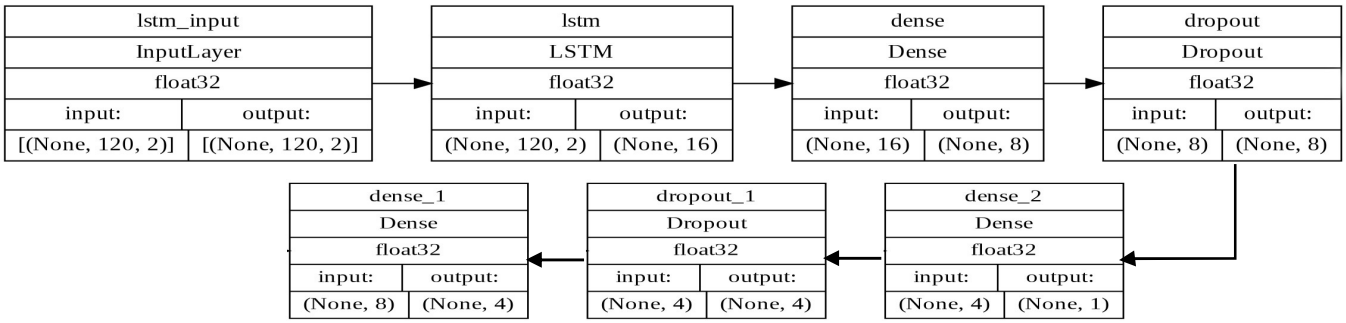

Figure 3: LSTM network architecture

B. *Feature Extraction*

Feature extraction transforms the raw HTTP access log files to a form suitable to be input to the LSTM network. Since HTTP requests have a small number of fields, the features are manually selected. The fields of the HTTP access logs are dependent on configuration but the request time is always present. We extract the time of request and the rps (request per second) for each request from an IP address. This results in two features per HTTP request of the form –

[time of request at time t in seconds, rps of the IP address at time t]

For each IP address, for each HTTP request, the tuple *t* (time, rps) is calculated. So, for each IP address, we get a sequence of tuple *t*. The value of tuple *t* is used to differentiate between a legitimate request and a recon scan. The LSTM network learns the sequences of tuple t for a legitimate user as well as for an attacker using a reconnaissance scan. This sequence of tuple *t* is fed as input to the LSTM network. The maximum length of a sequence for one IP address is 120. In case the maximum length of a sequence is less than 120, padding is used to make the length equal to 120. So the LSTM input consists of sequences of tuple *t* for all the IP addresses. Algorithm 2 is used for feature extraction from the raw HTTP access log files. In the proposed approach, the log files described in section IV (A) is processed using Algorithm 2 and it was converted to sequences of tuple *t* (time, rps). The training data set on feature extraction resulted in 24000 unique IP addresses. The test data set after feature extraction resulted in a total of 29000 unique IP addresses. Table 3 shows the number of IP addresses used for training and testing the model. The feature extraction hence results in 24000 unique sequences for training and 29900 unique sequences for testing.

Algorithm 2: Access logs to sequences

input ← Filtered HTTP access log files
output ← Sequences of HTTP requests grouped by IP Address

```
sequences[] = array()
for each file in input:
    lines = split(input, "\n" )
    ip_addresses = unique_ip(lines)
    for each IP in ip_addresses:
        for each request:
            time = get_request_time(request)
            rps  = calulateRpsForEachSecond(IP)
            sequence[] = (time,rps)
        endfor
    end for
end for
return sequences
```

Table 3: Data set after feature extraction

Data set	Name	Class	Number of Samples
Train	TS	Positive	12138
		Negative	12138
Test	DS1	Positive	3290
		Negative	3290
	DS2	Positive	5746
		Negative	5746
	DS3	Positive	5914
		Negative	5914

C. *Training*

a) LSTM Networks

LSTM networks are widely used in deep learning for learning from time-series data[12]. LSTM solves the problem of exploding and gradient disappearance and thereby can learn long term dependencies better than recurrent neural networks. An LSTM network is a neural network with a sequence of repeating modules. The repeating structure of neural networks is what enables the LSTM to remember long term dependencies present in the data.

The architecture of the proposed system using the LSTM network is shown in Figure 3 and is composed of 7 layers. The first layer is the input layer while the second layer is a 16-unit LSTM which takes as input sequences of length 120 and each sequence has two features, rps for that IP address for one time instant which is a second time instant of that request. The output of the first LSTM layer is fed to a dense layer with 16 neuron units. A dropout layer follows the dense layer for regularization with a value of 0.2. The next layer is the dense layer with 4 units followed by a dropout layer with a dropout value of again 0.2. The final layer is again a dense layer with a single unit and a sigmoid activation function. The output of the final sigmoid neuron layer is a probability value in the range 0.0 to 1.0. All samples whose probability value is greater than 0.5 is considered a reconnaissance attack and all samples whose probability value is less than or equal to 0.5 are considered a benign request.

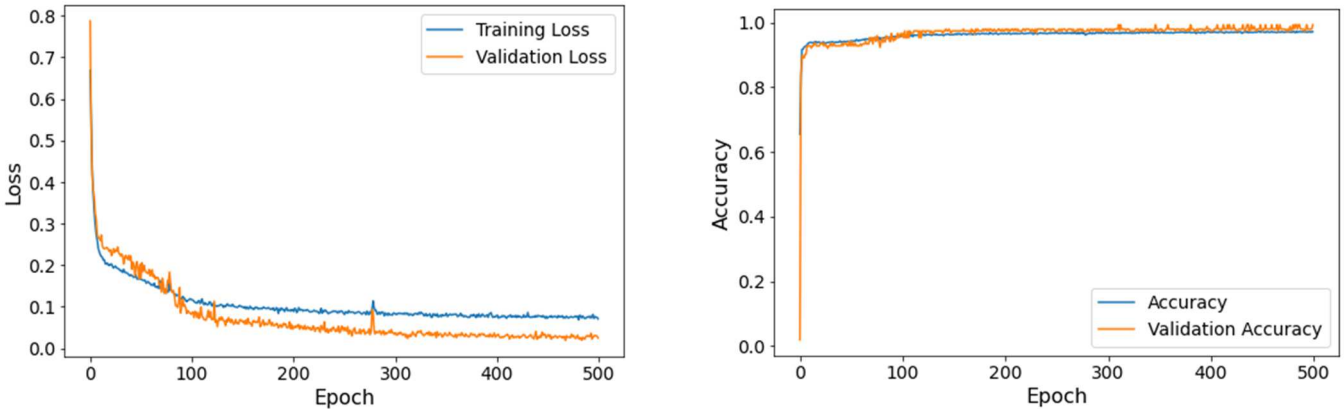

Figure 4: Accuracy and Loss during Training and Validation.

The training is controlled by hyper parameters which are user-specified values and cannot be learned by the network automatically. The various hyper parameters used in the proposed approach are shown in Table 4. The training performance of the model is measured in terms of training accuracy, training loss, validation accuracy and validation loss. The training accuracy is accuracy during training and validation accuracy is the accuracy achieved using the validation data set that is used for validating the trained model after each epoch. The training loss shows the loss during training while the validation loss shows the loss during validation after each epoch. The graph of training loss vs validation loss and training accuracy vs validation accuracy is shown in Figure 4. From Figure 4 it is seen that the training and validation loss decreases rapidly in the beginning and later settles after around 400 epochs. The gap between the training loss and validation loss is less than 0.1 and hence it can be concluded that the model is not overfitting or underfitting. The training accuracy and the validation accuracy also settle to around 0.9 after 400 epochs. The gap between validation and training accuracy is also less than 0.05 and indicates no apparent overfitting or underfitting. The accuracy may further increase with the increase in the number of epochs. Following achieving the desired training and validation accuracy as a result of hyper parameter tuning, the model is saved so that it can be evaluated on the test data set. The LSTM based model was implemented using Keras[8] and Tensorflow[9].

Table 4: Hyper parameter used in training

Hyper parameter	Grid search Result
Batch Size	256
Number of Epochs	500
Dropout Regularization	0.2
Learning Rate	0.0001
Beta	0.9
Number of neurons in hidden Layer 2,3,5	16,8,4
Optimizer	Adam
Loss Function	Binary Cross Entropy

V. EVALUATION

The selected model was evaluated on three test data set DS1, DS2, and DS3 and there is no overlap between the test data sets and the training data set. This was done to prevent any leakage of data from the test dataset to the training data set which might have resulted in overfitting in the model. The number of samples in each dataset is shown in Table 3. For evaluation of performance, the proposed method uses precision, recall and accuracy as metrics which are defined in terms of TP (True Positive), TN (True Negative,) FP (False Positive), and FN (False Negative). The confusion matrix, which lists the number of TP, FP, TN, and FP in numerical values, is shown in Figure 6. The confusion matrix is based on the combined dataset of DS1, DS2 and DS3. It shows that the model misclassified 368 samples out of 14582 as Negative and misclassified 515 samples as positive out of 14435. From the confusion matrix, it can be calculated the method achieves a mean accuracy of 0.97, precision of 0.96, and recall of 0.96 across the three test data set DS1, DS2, and DS3.

The performance of the model is also measured in terms of the ROC-AUC curve. The ROC curve is a probability curve that plots TPR (True Positive Rate) along the y-axis and the FPR (False Positive Rate) along the x-axis. The TPR and FPR are calculated as follows-

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{TN + FP}$$

The AUC (area under the curve) of the ROC gives the performance of the model and the higher the AUC, the better the performance. The AUC-ROC curve of the model against the three datasets DS1, DS2 and DS3 is shown in Figure 5. The AUC achieved for DS1, DS2 and DS3 are 0.99.

VI. CONCLUSION

In this paper, an approach to detect reconnaissance scans against web applications was proposed. Many approaches exist to detect reconnaissance scans like port scans that occur at the

8. https://keras.io/
9. https://www.tensorflow.org/

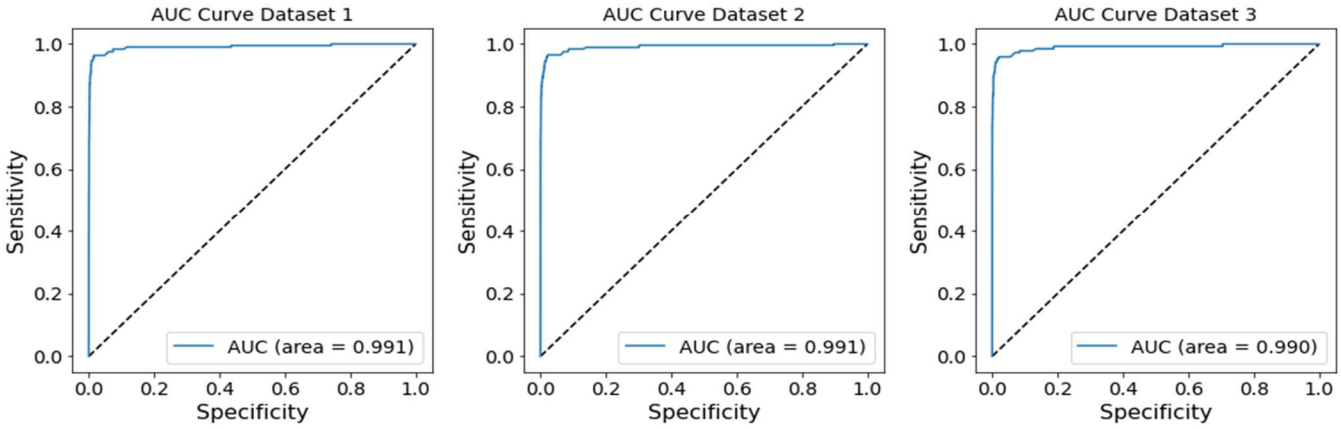

Figure 5: AUC-ROC for datasets DS1, DS2, and DS3.

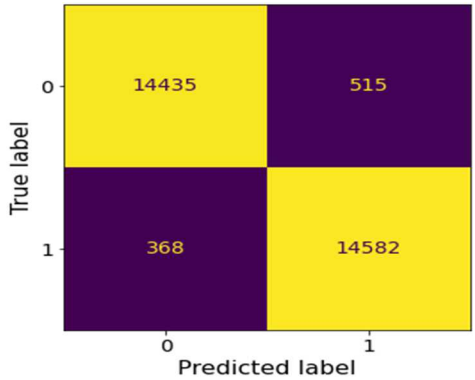

Figure 6: Confusion matrix for combined dataset.

application layer but there was very limited literature found that addresses the issue of reconnaissance scans against web applications. This paper, focused on detecting reconnaissance scans through analysis of web server access. The results obtained from the evaluation of the proposed approach indicated that the methods achieve excellent detection performance and can detect 99 percent of the reconnaissance scans. The false positive rate is also within 0.1. The proposed system does not work in real time, but it can also be extended for real time detection of reconnaissance scans. Overall, the proposed method can improve security of web applications by alerting administrators and other stakeholders whenever a reconnaissance scan is detected so that preventive or corrective measures can be deployed against any future exploitation attempts. In future, the proposed method can be extended to work with streaming data so that web reconnaissance scans can be detected as soon as it happens.

REFERENCES

[1] T. Threepak en A. Watcharapupong, "Web attack detection using entropy-based analysis", International Conference on Information Networking, no 3, bll 244–247, 2014.

[2] M. Zolotukhin, T. Hämäläinen, T. Kokkonen, en J. Siltanen, "Analysis of HTTP requests for anomaly detection of web attacks", Proceedings - 2014 World Ubiquitous Science Congress: 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, DASC 2014, bll 406–411, 2014.

[3] A. Juvonen, T. Sipola, en T. Hämäläinen, "Online anomaly detection using dimensionality reduction techniques for HTTP log analysis", Computer Networks, vol 91, bll 46–56, 2015.

[4] M. Moh, S. Pininti, S. Doddapaneni, en T. S. Moh, "Detecting Web Attacks Using Multi-stage Log Analysis", Proceedings - 6th International Advanced Computing Conference, IACC 2016, bll 733–738, 2016.

[5] Q. Cao, Y. Qiao, en Z. Lyu, "Machine learning to detect anomalies in web log analysis", 2017 3rd IEEE International Conference on Computer and Communications, ICCC 2017, vol 2018-Janua, bll 519–523, 2018.

[6] M. Baş Seyyar, F. Ö. Çatak, en E. Gül, "Detection of attack-targeted scans from the Apache HTTP Server access logs", Applied Computing and Informatics, vol 14, no 1. bll 28–36, 2018.

[7] Z. Zhang en C. N. Manikopoulos, "Architecture of the Reconnaissance Intrusion Detection System (RIDS)", Proceedings fron the Fifth Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC, bll 187–194, 2004.

[8] H. U. Baig en F. Kamran, "Detection of port and network scan using time independent feature set", ISI 2007: 2007 IEEE Intelligence and Security Informatics, bll 180–184, 2007.

[9] B. Hartpence and A. Kwasinski, "Combating TCP Port Scan Attacks Using Sequential Neural Networks", 2020 International Conference on Computing, Networking and Communications, ICNC 2020, bll 256–260, 2020.

[10] J. Wang, M. Zhang, X. Yang, K. Long, en J. Xu, "HTTP-sCAN: Detecting HTTP-flooding attack by modeling multi-features of web browsing behavior from noisy web-logs", China Communications, vol 12, no 2, bll 118–128, 2015.

[11] N. Chawla, K. Bowyer, L. Hall, en W. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", J. Artif. Intell. Res. (JAIR), vol 16, bll 321–357, 06 2002.

[12] S. Hochreiter en J. Schmidhuber, "Long Short-term Memory", Neural computation, vol 9, bll 1735–1780, 12 1997.