Journal of Network and Computer Applications 198 (2022) 103270

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling

Waleed Bin Shahid [a], Baber Aslam [a], Haider Abbas [a,*], Saad Bin Khalid [a], Hammad Afzal [b]

[a] Department of Information Security, National University of Sciences and Technology (NUST), Islamabad, Pakistan
[b] Department of Computer Software Engineering, National University of Sciences and Technology (NUST), Islamabad, Pakistan

ARTICLE INFO

Keywords:
Web Security
Web Application Security
Deep learning
Attacker profiling
Deception
Cookies
HTTP

ABSTRACT

Protecting web applications is becoming challenging every passing day, primarily because of attack sophistication, omnipresence of web applications and over-reliance on traditional Web Application Firewalls (WAFs). Advanced Persistent Threats (APTs) make overwhelming use of web attacks during infiltration and expansion phase. Noteworthy research has been carried out to detect web attacks using deep learning because traditional approaches fail against complicated attacks having crafted payloads, scripts and cookie manipulations. This paper proposes a framework based on an enhanced hybrid approach where Deep Learning model is nested with a Cookie Analysis Engine for web attacks detection, mitigation and attacker profiling in real time. We first generated a huge dataset over a period of time and trained our Convolution Neural Network (CNN) based deep learning model using Hypertext Transfer Protocol (HTTP) request parameters like Type, Content length, Data and Requested URL etc. We also developed a Cookie Analysis Engine that checks all incoming cookie(s) for integrity, mutations and failed sanitization checks and informs the user about probable privacy infringement by third party cookies. The framework analyzes the cascading output from the classifier and cookie analysis engine and takes the final decision. We performed rigorous testing of the proposed framework wherein it was first validated on our own custom dataset giving an accuracy of 99.94%. It was also validated on a publicly available benchmark dataset and gave an accuracy of 98.74%. When deployed in a controlled real time environment, the attacker profiling feature enabled the framework to save useful processing time as the deep learning classifier does not get triggered for every incoming request. This makes it easy to deploy in any environment to protect web applications in real time.

1. Introduction

The ubiquity of internet has resulted in a momentous growth in its usage all across the globe resulting in roughly half of humanity having an internet connection (Max Roser and Ortiz-Ospina, 2020). This perpetual digital transformation in human lives has intrigued end users, corporates, small businesses, large organizations and in fact everyone to share their data and information via websites. The number of websites in the world have surpassed 1.7 Billion and are still growing (Liedke, 2020). These websites are connected to backend databases in real time where the later, although not visible or directly accessible to the user are updated, parsed and accessed by the web application(s) as a result of some user activity (Kandpal et al., 2021). Besides catalyzing this massive e-transformation, the overwhelming use of webpages and reliance on websites has also stirred the attackers to compromise these sites in order to get hold of important data that might be of some financial or reputational gain. According to 2019 Verizon Data Breach Investigations Report, web attacks are the most prevalent sort of cyber-attacks, happening once every 39 s (Cukier, 2007; Paine, 2019). These attacks often result in massive data, financial and reputational loss and are further aggravated due to the absence of cybercrime regulations in many parts of the world.

Security experts have developed ways and means to tackle the grave threat of chronic web attacks. On the other hand, attackers have equipped themselves with more sophisticated approaches to circumvent these defensive mechanisms (Hamam and Derhab, 2021). OWASP (Open Web Application Security Project) has listed the top 10 web attacks (The Open Web Application Security Project, 2017) that pose a great deal of challenge to security experts (Shoel et al., 2018). These attacks have accentuated the need to not only go for secure coding in web applications (Meng et al., 2018) but to also rely on security mechanisms that can help thwart such attacks in real time.

* Corresponding author.
E-mail address: dr.h.abbas@ieee.org (H. Abbas).

https://doi.org/10.1016/j.jnca.2021.103270
Received 22 January 2021; Received in revised form 7 June 2021; Accepted 30 October 2021
Available online 14 November 2021

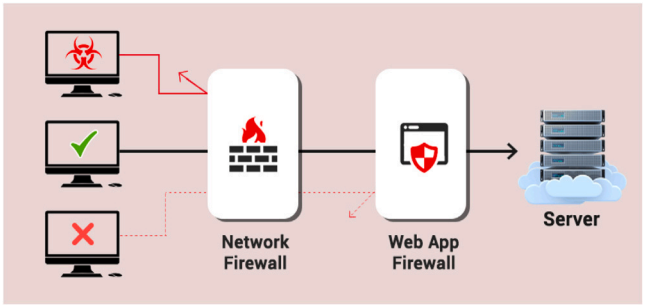

Fig. 1. Web Application Firewall (Nair, 2019).

One obvious choice is to use a WAF (Web Application Firewall (Clincy and Shahriar, 2018)), as shown in Fig. 1, that acts as a layer of defense standing between the attacker and the website. WAFs analyze user requests in order to filter the illicit traffic out of the routine one. This analysis of the HTTP requests (Wittern et al., 2017) is based on a set of predefined rules designed to discriminate the malicious traffic.

One of the most effective means to assess the security of web applications is to make use of WAVs (Web Application Vulnerability Scanners) (Sarkar, 2021). These tools analyze web applications against common web related vulnerabilities and launch all sorts of attack packets using their built-in malicious payloads to compromise the target website's confidentiality, integrity or availability (Bitzer et al., 2021). WAFs perform better against these scanners because of known signatures and payloads, but often fail against experienced hackers who prefer writing their own pieces of code. Even the most popular WAFs such as ModSecurity (Trustwave, 2020) fails in detecting such HTTP requests. These issues with WAFs are either because of misconfigurations, their rule-based nature, lack of semantic analysis and behavioral detection, failure against permutation of known attacks and no protection against zero-day attacks . Moreover, tired of false positives, many security administrators keep their WAFs on the *Alert Mode*, in which attacks are never blocked in real time. According to a survey conducted by Ponemon institute (Vicente, 2019), only 9 percent of the WAF users say that their solutions have never been compromised, whereas, 65 percent complain that attackers can bypass their solutions without much of an effort.

These immense problems associated with traditional rule-based WAFs accentuate the need for incorporating machine learning for automated real-time attack detection. Many research works use traditional machine learning classifiers like Naive Bayesian, Decision Tree, K-NN and SVM (Betarte et al., 2018; Dong and Zhang, 2017; Althubiti et al., 2017; Zhou and Wang, 2019), but these approaches fail to provide better accuracy and precision against unknown data. Deep Learning based approaches have gained much importance as they outperform traditional machine learning classifiers in terms of accuracy and precision but there are certain issues associated with the use of deep learning based methods while dealing with web attacks. Firstly, their strength depends on the data they are trained on and are unable to give same performance on anything that appears different and would require re-training. Many research works as discussed in Section 2 have not cross-validated their proposed techniques either on publicly available benchmark datasets or by real-time deployment. Another major issue with deep learning methods is the huge processing power they require. Almost all related research works ignore this issue and route every incoming HTTP request to the deep learning classifier. This phenomenon incurs a lot of latency and performance decadence in real-time systems.

Therefore, development of any web attack detection system must consider achieving certain security goals. Firstly, the system should have very high accuracy and precision values. This would only happen if it is based on deep learning and not traditional classifiers or rule-based techniques. Performance and efficiency are other key security goals because they would make any detection system suitable for real-time deployment. Since applying deep learning would incur a lot of performance decadence and latency, therefore, smarter techniques need to be adopted which help in optimizing attack detection. Validation is another vital security goal which would guarantee any attack detection system's performance on unknown data.

The proposed web attack detection framework achieves all these security goals by addressing all performance and optimization issues related to Deep Learning based classifiers. The main research contributions of this research work are listed as follows:

1. A novel framework has been proposed which is based on a hybrid approach which nests Deep Learning model with a Cookie Analysis Engine for web attacks detection, mitigation and attacker profiling in real time.
2. The Convolutional Neural Network based deep learning classifier is trained using HTTP request parameters like Content Type, Length, Requested URL and Data etc. on a large dataset specifically generated for this purpose.
3. A Cookie Analysis Engine that has been designed to check all incoming cookie(s) for integrity failures, mutations and failed sanitization checks and informs the user about probable privacy infringement by third party cookies.
4. An efficient framework which saves useful processing time when deployed in real time as the attacker profiling feature limits the execution of deep learning classifier for every incoming HTTP request without degrading attack detection capability. The proposed framework gives an accuracy of 99.94% on our testing dataset and 98.74% on a publicly available benchmark dataset.

This paper is divided into five sections. Introduction section is followed by Section 2 which consists of relevant research works related to detection and mitigation of web attacks. These techniques have been compared and evaluated on the basis of their strengths and weaknesses which necessitates the idea to perform further research in this area. Section 3 discusses dataset generation comprising of both benign and malicious HTTP requests, along with the capturing of POST HTTP requests. The proposed framework comprising of the Cookie Analysis Engine and the Deep Learning classifier has been discussed in Section 3.4.1 and Section 3.4.2 respectively. Section 4 provides the complete testing, performance evaluation, results and comparative analysis of the proposed framework with related research works. The research work is concluded in Section 5 where future work related to development of a web deception system has also been discussed.

2. Related work

Several research works have discussed the use of deep learning based solutions for detecting and mitigating web attacks and scanning probes.

Luo et al. (2020) proposed an ensemble classification model based on three deep learning models where the final decision is based on results extracted out of the three classifiers. The research gives high accuracy and low false positive values. Niu and Li (2020) proposed a web attacks detection framework by combining Convolutional Neural Network (CNN) with Gated Recurrent Unit (GRU) approach. This model gives high accuracy. Both research works made use of the CSIC data (Giménez et al., 2012) for training and testing their models. Kim and Cho (2018) presented a C-LSTM neural network based approach to model incoming web traffic. They demonstrated that C-LSTM analyzes the web traffic deeply as it combines both CNN and LSTM to extract temporal and spatial features of web data. The proposed work achieves an accuracy of 98.6% and a recall value of 89.7% on Yahoo's Webscope S5 dataset. Liang et al. (2017) make use of Recurrent Neural Network

(RNN) to train their model only on normal HTTP GET requests from the CSIC dataset (Giménez et al., 2012) and malicious HTTP GET requests from ModSecurity (Trustwave, 2020) logs. The proposed methodology yields an accuracy of 98.42% in comparison to ModSecurity. These proposed approaches have nested different deep learning models in order to achieve higher accuracy but at the cost of processing power as they trigger all models for every incoming HTTP request thereby incurring latency and computational issues when protecting web applications in real time.

Tekerek (2021) also presented a CNN based approach for detection of web attacks with high accuracy on CSIC dataset (Giménez et al., 2012). Pan et al. (2019) came up with an end-to-end deep learning approach for detecting SQLi, XSS and deserialization attacks and evaluated their model on a synthetic dataset as well as production applications with known vulnerabilities. Mahfoudhi (2021) and Jemal et al. (2021) presented a CNN based code level and ASCII embedding technique respectively. The later detects web server attacks with an accuracy of 98.24% on CSIC dataset (Giménez et al., 2012). Mokbal et al. (2019) presented an artificial neural network based on multi-layer perceptron approach (MLP) for detecting XSS attacks by dynamically extracting features from web traffic. It achieved an accuracy of 98.97%. Liu et al. (2019) presented a Locate-Then-Detect (LTD) system for detecting malicious payloads exploiting XSS and SQLi attacks by using attention based deep neural networks. The proposed work uses limited features for training purpose and yields good performance on both CSIC (Giménez et al., 2012) and real-world data.

Fang et al. (2018) presented an LSTM based deep learning based approach to counter XSS attacks. It was tested on real-time dataset with an accuracy of 99.98%. Abaimov and Bianchi (2019) detected code injection attacks like SQLi and XSS using convolutional deep neural network resulting in high levels of accuracy on real-world datasets. The proposed research work only uses operators, escape symbols and expressions as features for training the CNN which is not sufficient for detecting advanced payloads. Although few of these proposed research works have generated their custom datasets for classifier training but most of them have made use of the same publicly available dataset for both training and testing. Moreover, these proposed approaches only address limited number of attacks like XXS and SQLi and do not detect all of the OWASP top ten attacks.

Zhang et al. (2017) proposed a CNN based technique for detecting SQLi, buffer overflow, CRLF injection, XSS and parameter tampering attacks. The system was not trained or tested on real-time data but rather relied on CSIC dataset (Giménez et al., 2012). The proposed approach studies the HTTP requests to form word sequences for the data pre-processing stage. It only detects web attacks hidden in URLs and does not analyze the complete HTTP request message. The framework achieves the accuracy of 96.49% after 10 epochs of training which is not efficient enough for deployment and testing in real time. Tian et al. (2020) proposed a deep-learning based mechanism for detection of web attacks from the analysis of URLs. It also makes use of word2vec (Wen et al., 2016) in order to look for words in the URL. Since the proposed approach only caters for detecting attacks from URLs, it does not detect web attacks based on HTTP POST requests. The proposed work has a detection accuracy of 99.41%.

Some research works have also added behavioral detection in WAFs to enhance their functionality. Moradi Vartouni et al. (2019) presented an anomaly based WAF using deep neural networks. The system gives an accuracy of 89.24% when tested on CSIC (Giménez et al., 2012) dataset. Khan et al. (2017) presented a machine learning driven approach to detect malicious JavaScripts based on an interceptor designed to protect the client side as it performs static analysis on server's response to client's HTTP GET request for investigating potential attacks in the source code to be executed by the client-side browser. A major drawback of the proposed approach is that it would fail to detect zero-day attacks as behavioral analysis of malicious code has not been taken care off. It achieves an accuracy of 97.14% by using the KNN approach. Birkinshaw et al. (2019) presented an SDN (Software Defined Networking) based approach that counters port scanning and DDoS attacks using Reinforcement Learning algorithm (Recht, 2019). The proposed work does not address a wider range of attacks, specifically the top web attacks.

Dong and Zhang (2017) presented an adaptive learning based approach, named SVM hybrid to tackle malicious queries in web attacks. They use an ensemble classifier based on meta-learning for underlying detection of malicious queries and outperform other malicious query detection approaches in performance. The proposed technique achieves an F score of 94.79%.

Few research works have particularly focused on detecting WAVs and web crawlers instead of attack detection. Stevanovic et al. (2011) came up with a couple of features in order to detect web crawlers using data mining classification algorithms. The research makes use of Sequential Request Ratio and Standard Deviation of page requests in order to differentiate web crawlers from normal users accessing a university website. The proposed scheme appears to stand weak against modern web scanners which are good at mimicking the behavior of normal human users. Moreover, the research work only targets web crawlers and does not address scans probing against any of the OWASP top 10 vulnerabilities. Dong and Zhang (2017) presented an adaptive learning based approach, named SVM hybrid to tackle malicious queries in web attacks. They use an ensemble classifier based on meta-learning for underlying detection of malicious queries and outperform other malicious query detection approaches in performance. The proposed technique achieves an F score of 94.79%.

Yang et al. (2019) proposed a hierarchical correlation-based web application scan detection technique to figure out coordinated attacks from branded malicious sources. The model makes use of semantic correlation analysis and temporal–spatial correlation analysis to distinguish the coordinated web scanners. Later, Bro/Zeek network monitor is used to collect web traffic logs from some web hosting service provider and the proposed scheme is evaluated on those logs. The system achieves 97% accuracy. Zhang et al. (2019) presented an approach that assimilates attack indications from web requests and their associated responses in order to detect XSS vulnerabilities with an accuracy of 93.98%. The proposed model extracts attack features from existing attack datasets using the word2vec (Wen et al., 2016) algorithm and compares them with normal traffic in order to train the GMM (Gaussian Mixture Model).

Table 1 contains a summary of related work that has been carried out as discussed in this section.

3. Proposed framework for web attacks detection mitigation and attacker profiling

The proposed framework detects and mitigates web attacks in real time along with attacker profiling. The incoming HTTP request is first examined by the profiler that is maintained and updated over time. If the user profile associated with the incoming request is benign and a static page has been requested, the page will be opened without triggering either the deep learning classifier or the cookie analysis engine. Moreover, requests by malicious users will be blocked straight away while all other requests will be forwarded for further analysis. This feature of the proposed framework improves efficiency and saves processing time. The request forwarded by the profiler is first passed through a bot and scanning detection engine which analyzes it with respect to time, requested URL and socket information of the requester. If a bot or scanner is detected, the request will be turned down and the user will be profiled malicious. Otherwise, the request will be split in a way that the cookie(s) are analyzed by the Cookie Analysis Engine while remaining HTTP parameters go to the deep learning classifier for further analysis. The final decision is based on their outcome, and the profiler is updated accordingly as shown in Fig. 2.

Table 1
Existing research works on using deep learning to detect web attacks.

Research Work.	Year	Technique	POST HTTP Analysis	Attacks Mitigated	Dataset used
Tekerek (2021)	2021	CNN	NA	Web Attacks	CSIC dataset (Giménez et al., 2012)
Mahfoudhi (2021)	2021	Code Embedding using CNN	NA	Basic Web Attacks	CSIC dataset (Giménez et al., 2012)
Luo et al. (2020)	2020	Ensemble approach (Deep Learning)	Yes	Web Attacks	CSIC dataset (Giménez et al., 2012)
Niu and Li (2020)	2020	CNN + GRU	Yes	Web Attacks	CSIC dataset (Giménez et al., 2012)
Mokbal et al. (2019)	2019	ANN based MLP	Yes	XSS	Custom Dataset
Yang et al. (2019)	2019	Hierarchical correlation, Temporal–spatial correlation	No	Web scans	Real-world dataset
Moradi Vartouni et al. (2019)	2019	Deep Neural Network	No	OWASP Top-10	CSIC dataset (Giménez et al., 2012), PKDD2007
Abaimov and Bianchi (2019)	2019	CNN	No	SQLi, XSS	Real-time dataset
Liu et al. (2019)	2019	Attention based DNN	No	SQLi, XSS	CSIC dataset (Giménez et al., 2012), Real-time dataset
Zhou and Wang (2019)	2019	Ensemble learning approach	Yes	XSS attacks	Real-world dataset
Tian et al. (2020)	2019	Deep learning	No	OWASP Top-10	CSIC dataset (Giménez et al., 2012), FWAF, HttpParams Dataset
Kim and Cho (2018)	2018	CNN + LSTM	NA	All	CSIC dataset (Giménez et al., 2012)
Fang et al. (2018)	2018	word2vec, LSTM	Yes	XSS	Real-time dataset
Liang et al. (2017)	2017	RNN	No	OWASP Top-10	CSIC dataset (Giménez et al., 2012)
Zhang et al. (2017)	2017	CNN	No	SQLi, Buffer overflow, CRLFi, XSS	CSIC dataset (Giménez et al., 2012)

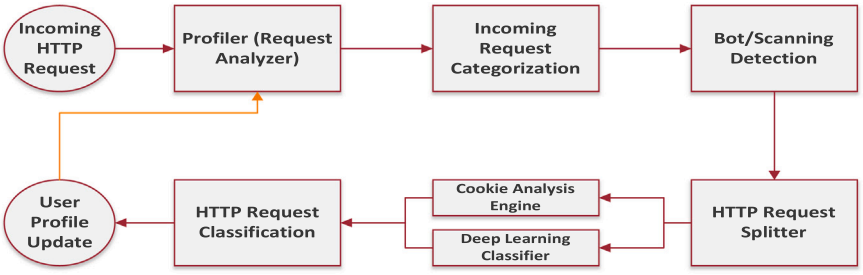

Fig. 2. Block diagram of the proposed framework.

3.1. Creation of dataset

For generating our dataset we set up the environment by deploying a vulnerable PHP/MySQL web application that contained pages susceptible to a large variety of web attacks including the OWASP top ten. We first targeted our website with a large number of custom attack scripts and payloads for SQLi, XSS, CSRF, XXE, directory traversal and broken authentication etc. along with their permutations. Apart from that, we used open-source WAVs like OWASP-ZAP (OWASP, 2020), Nikto (Chris Sullo, 2020) and Arachni (Arachni, 2020) and a commercially licensed scanner Burpsuite (Portswigger Web Security, 2020) for generating attack traffic. It was made sure that no benign traffic was targeted towards the website during this period. This traffic was later logged and parsed as shown in Fig. 3 for later use.

For generating benign dataset, the target website having all features and contents of a normal real world website was accessed over a period of one month from various machines at different intervals and the logs were captured as shown in Fig. 3. As expected, these logs had no abnormality in any of the header fields. Unlike the attack packets, benign requests only had GET and POST request types, web browser as the user-agent and accessed limited URLs in given time, unlike many of the attack packets.

Our dataset contains a large number of attacks as compared to publicly available benchmark datasets. These were generated by using crafted payloads, their mutations and different vulnerability scanners. Our dataset also contains a larger number of benign HTTP requests.

In our case, the number of benign HTTP samples exceeded their malicious counterparts. We therefore used SMOTE (Chawla et al., 2002) over-sampling technique for balancing both class labels. Malicious samples were augmented by selecting those which are relatively closer in the feature space. A line was then drawn between these selected samples in the feature space and additional samples were synthetically created along that line. For instance, let $\boldsymbol{M} = (m_1, m_2, m_3, \ldots.., m_n)$ be the set of all malicious HTTP requests generated, whereas n are the total number of samples belonging to the malicious class.

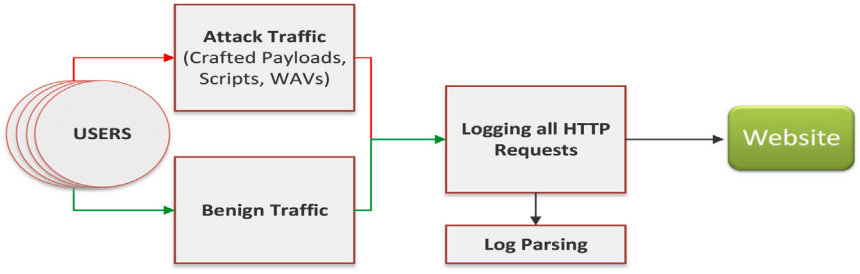

Fig. 3. Process of malicious and benign logs generation.

Algorithm 1 Class Balancing using SMOTE

Input : $\boldsymbol{M}=(m_1, m_2, m_3,\ldots..., m_n)$
Output : M'

1: Step-1: Compute $\boldsymbol{K}$: *k-nearest neighbour* ← Euclidean distance of m_i with all in $\boldsymbol{M}$
2: Step-2: Set sampling rate $\boldsymbol{S}$ based on imbalanced proportion
3: Step-3: For each m_i in $\boldsymbol{M}$: $\boldsymbol{S}$ samples are picked from $\boldsymbol{K}$ to create M'
4: Step-4: For each m'_i in M': compute new M'_i such that m'$_n$ ← (m' + *rand(0,1)*)*(m' - m$_k$)

3.2. Analysis of attack logs

Malicious logs reveal key information about various attack methodologies, crafted scripts, payloads and WAVs. For instance, attackers launch spidering probes on the website to enumerate as much as they can for exploitation purpose. The attack logs happen to be so random that a certain opinion can never be formed about their malicious intent by simply analyzing the request parameters in detail. Fig. 4 shows detail of HTTP Request Type parameter in the attack dataset. Moreover, attackers often tamper verbs in the HTTP Request field in order to analyze the generated response. Scanners also conceal their name so that the target application believes the request to be coming from a normal browser. Attackers can also leave this field blank in order to analyze the generated response. Details of how the Request Type and User Agent fields are forged in our attack dataset are shown in Fig. 5.

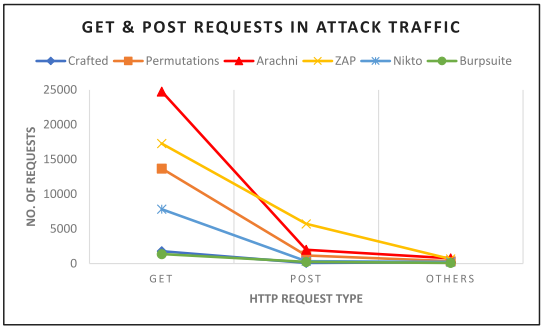

Fig. 4. Details of HTTP Request parameter.

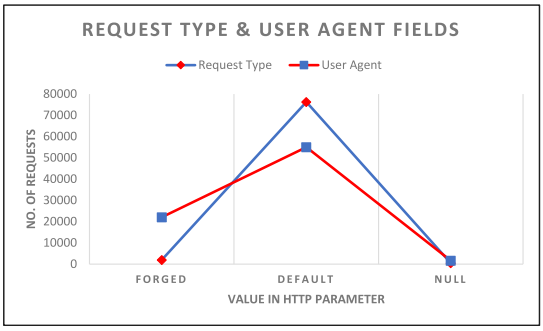

Fig. 5. Details of User Agent and Request Type.

3.3. Bot and scanner detection

Novice attackers prefer using bots and popular scanners to attack a website as they often bombard a particular URL with multiple requests in a very short time span, whereas normal users take some time on web pages and respond with appropriate headers and parameters to move with the website flow. The proposed bot and scanner detection module deals with such HTTP requests by queuing up a few packets and analyze them on the basis of time, requested URL and sender's identity and socket information as explained in algorithm 2. If found malicious, the request is dropped and feedback is sent to the profiler who marks the requester as a malicious user. Otherwise, the request is forwarded to the cookie analysis engine and deep learning classifier.

Algorithm 2 Bot and Scanner Detection Algorithm

Input : HTTP Requests
Output : blacklist source
Initialize : counter ← 0 , limit ← 3 , i ← 2 , t$_a$, t_i , t_n , S_i , S_n , D_i , D_n

1: **while** i > 0 **do**
2: **if** $S_n = S_i$ and $D_n = D_i$ **then**
3: compute(Δt$_{i,n}$);
4: **if** Δt$_{i,n}$ < t$_a$ **then**
5: counter ← counter+1;
6: **end if**
7: **end if**
8: **if** counter *equals* limit **then**
9: blacklist(S_n)
10: **end if**
11: i ← i-1;
12: **end while**

3.4. Classifier training

From an attack point of view, all fields of an incoming HTTP request are inter-related and, neither in isolation nor in any combination, are they helpful in determining the type and intent of the request. For instance, the HTTP request shown in Fig. 7 appears benign as it is a simple GET request with no injection in language or encoding fields, no cookie manipulation, and a simple requested URL. Any rule based security application will label this HTTP request as benign but in fact it is not because it is a type of directory bruteforcing attack where the attacker is checking for */operador* directory, as highlighted. ModSecurity (Trustwave, 2020) lets this HTTP request pass safely.

Similarly, Fig. 8 looks malicious because any rule-based security solution will block IP addresses in user submissions, as highlighted but in this special case, it is required by the application itself and is not malicious in any way. This request is also blocked by ModSecurity (Trustwave, 2020). Experienced attackers can misuse any parameter of the HTTP request like *Type*, *User Agent*, *Data* or even *Language* as explained in Table 2. Cookie mutations and injections are also important but are specific to the target website. It is hard to determine as to which HTTP parameter alone helps in determining about the nature of request. Therefore, in this research, we split the incoming HTTP request so that the cookie(s) go to the Cookie Analysis Engine while the remaining parameters of HTTP request go the deep learning classifier.

3.4.1. Cookie(s) analysis engine

The Cookie Analysis Engine as shown in Fig. 6 takes all cookies of an HTTP request as input. CAE would first detect presence of third party cookies and notify the user about probable privacy infringement as these tracking cookies are mostly used by advertisers (Gomer et al., 2013). It would then launch integrity checks on local cookie(s) to see if they have been modified by the sender. If the integrity checks are successful the cookie(s) are marked as benign. Cookies where modifications are not malicious will also fail integrity checks. Therefore, such cookies along with all third party cookies (if present) are passed through sanitization checks aimed at finding unwanted characters, symbols, trackers, script parameters and values etc. in any field. Failed sanitization checks for any cookie as shown in Fig. 9 result in marking the incoming cookie(s) as malicious.

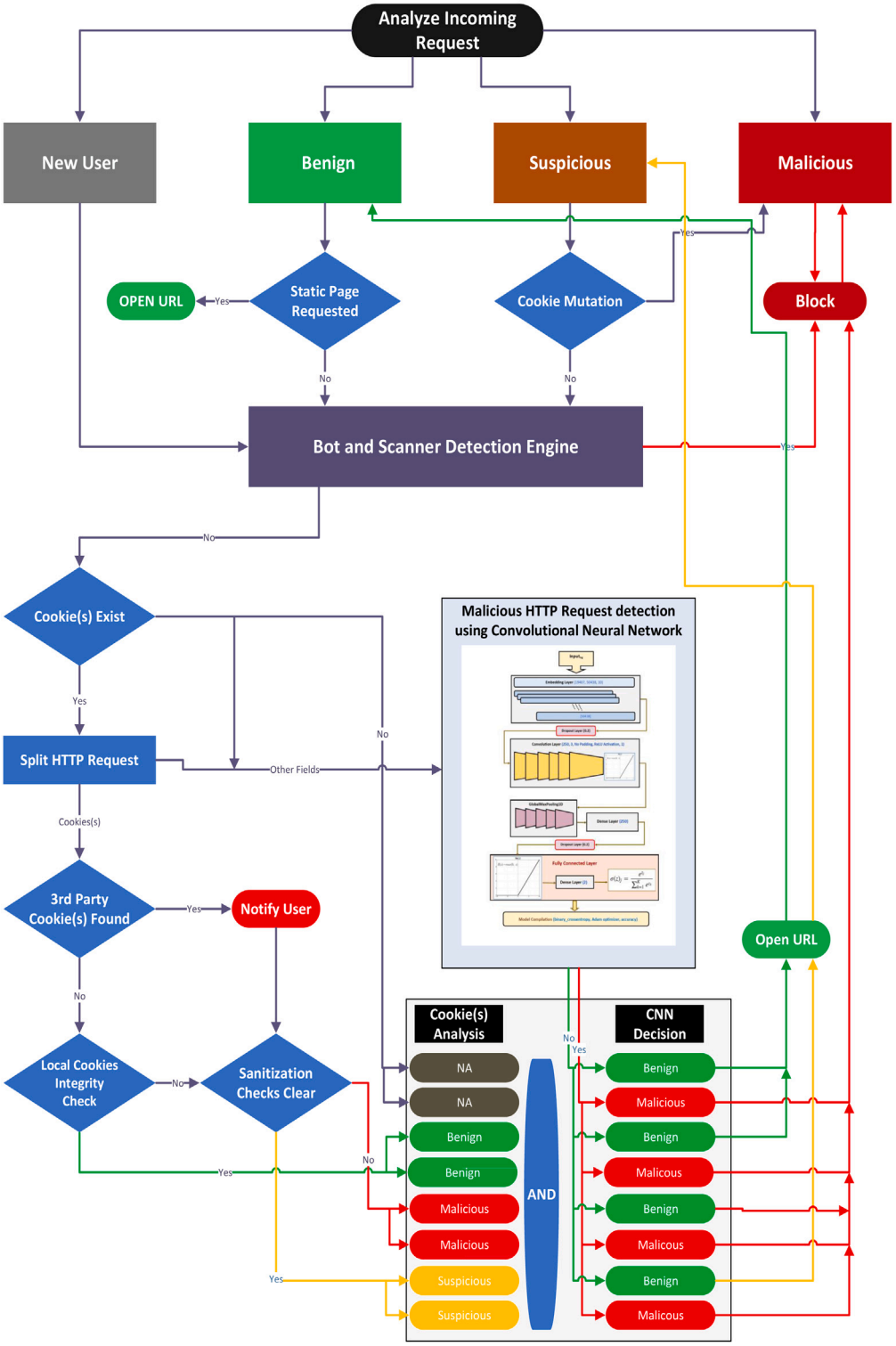

Fig. 6. Flowchart of the proposed attack detection, mitigation and attacker profiling framework.

```
# Wed Mar 25 02:32:40.131510 2020 - pid 12628 - client 192.168.8.104:45284
curl -v \
--header "User-agent:" --header "Accept:" \
--request "GET  " \
--header "Accept-Encoding: gzip,deflate" \
--header "User-Agent: Mozilla/5.0(WindowsNT10.0;Win64;x64;rv:71.0)Gecko/20100101Firefox/71.0"
--header "Accept: text/html,application/xhtml+xml,application/xml;q0.9,/;q0.8" \
--header "Accept-Language: en-US,en;q0.8,he;q0.6" \
--header "Cookie: PHPSESSID4494qtkdkef667tl02vmv9dsqv;securitylow" \
-http://192.168.8.104/DVWA/operador
```

Fig. 7. A Benign looking Malicious HTTP request.

```
# Wed Mar 25 03:11:51.186441 2020 - pid 12841 - client 192.168.8.104:51016
curl -v \
--header "User-agent:" --header "Accept:" \
--request "POST" \
--header "Accept-Encoding: gzip, deflate" \
--header "User-Agent: Mozilla/5.0(WindowsNT10.0;Win64;x64;rv:75.0)Gecko/20100101Firefox/75.0" \
--header "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/;q=0.8" \
--header "Accept-Language: en-US,en;q=0.5" \
--header "Cookie: PHPSESSID=iog8qjcdgo5q3o99i6kt26hd40;hibext_instdsigdipv2=1,
                  ip=192.168.100.1&Submit=Submit&user_token=562d2e7c5cf3fec46a70ac4083866f4b,74"
http://192.168.100.50/DVWA/vulnerabilities/exec/
```

Fig. 8. A Malicious looking Benign HTTP request.

```
GET /userinfo.php HTTP/1.1
Host: www.xssroks.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Accept-Language: en-GB,en;q=0.5
Referer: https://www.xssroks.com/
Connection: keep-alive
Cookie: SessionID_JAR=2020-12-10-16-JYTBS199823HSKZPQ; UID=206; Loc=Hawai<BODY
BACKGROUND="javascript:alert('XSS')">;
```

Fig. 9. Cookie with failed integrity and sanitization check.

```
POST /admin/updatecookie.php HTTP/1.1
Host: www.paratampering.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Accept-Language: en-GB,en;q=0.5
Referer: https://www.paratampering.com/
Connection: keep-alive
Cookie: SessionID_JAR=2020-12-10-16-JYTBS199823HSKZPQ; UID=1; Loc=admin;
```

Fig. 10. Harmful mutation in the cookie.

Table 2
HTTP features and their attack scenarios.

Features	Attack scenarios
Request Type	Attackers often change this field deliberately when requesting a particular resource by tampering HTTP verbs. For instance, *OPTIONS* request type helps in analyzing the response headers that will show all the allowed HTTP methods.
User Agent	This field is usually different in benign and attack traffic, as the later might reveal the scanner's name, if not forged or obfuscated. If the attacker has somehow forged this field as shown in Fig. 7, then this feature alone will fail to determine the attack as other features like detecting payload and timing analysis will serve the purpose.
Content Length	Content length in attack probes do not provide true representation of the data being sent. Few attack packets in our dataset had *0* as the content length value despite the fact that data was being sent. Similarly, few attack packets had large Content Length value with no data being sent in GET requests. This important feature alone provides limited help because the user application might not send content length in the first place.
URL	This parameter must also be looked upon in relation to sender's profile and authorization. Fig. 7 is a brilliant example of the fact that an HTTP request which looks perfectly benign is in fact malicious because of the directory brute-forcing attack, where the attacker requests a particular URL.

For instance, in Fig. 10, the *UID* and *Loc* are different from what was assigned to this *SessionID_JAR*. We consider the incoming cookie(s) to be suspicious, if sanitization checks for all of them are clear but the local website cookie(s) fails the integrity checks which means that the mutated cookie is not harmful for the website we want to protect.

This separate analysis of cookie(s) in the CAE also help us in settling somewhere between malicious and benign, we call suspicious. The details of how the Cookie Analysis Engine decides about the fate of incoming cookie(s) is mentioned in Table 3.

Table 3
Cookie analysis engine working.

Third Party Cookie(s) Found	x	x	x	✓	✓	✓	✓	✓	✓
Sanitization Checks ↑	–	–	–	✓	✓	✓	x	x	x
Local Cookies(s) Integrity Check	x	x	✓	✓	x	x	x	x	✓
Sanitization Checks ↑	✓	x	–	–	✓	x	x	✓	–
CAE Decision	S	M	B	S	S	M	M	M	M

3.4.2. Deep learning based classifier

Convolutional Neural Networks (CNNs) considerably enhance the efficiency of neural networks as the convolution operation performs matrix multiplication on small regions of data. In this research, we are dealing with HTTP request logs which are represented by a single sequence of multiple features. Therefore, two-dimensional CNN is not required. This would also result in low computational complexity as the forward and backward propagation only requires simple array operations. We have already consumed cookie(s) in the CAE, so the remaining HTTP request parameters are used for training the 1D-CNN. Firstly, the training dataset $\boldsymbol{D_{trg}}$ is fed into the system which is passed through layers of embedding, dropout, convolution, pooling and densing before ending up at the fully connected layer as shown in Fig. 11.

Our HTTP requests vector $\boldsymbol{V}$ can be represented as a set of features (f_1, f_2, ….., f_n), where $\boldsymbol{n}$ holds the number of possible values in the window, f_i represents the feature's value in normalized form, $\boldsymbol{i}$ represents the index of the feature value and $\boldsymbol{f_m}$ represents the index of feature map for each window of the input vector. We then move on to defining initializing parameters for our CNN as listed below:

1. $\boldsymbol{W}$ represents the maximum number of distinct words in the training dataset $\boldsymbol{D_{trg}}$
2. $\boldsymbol{S}$ represents the size of the embedding vectors

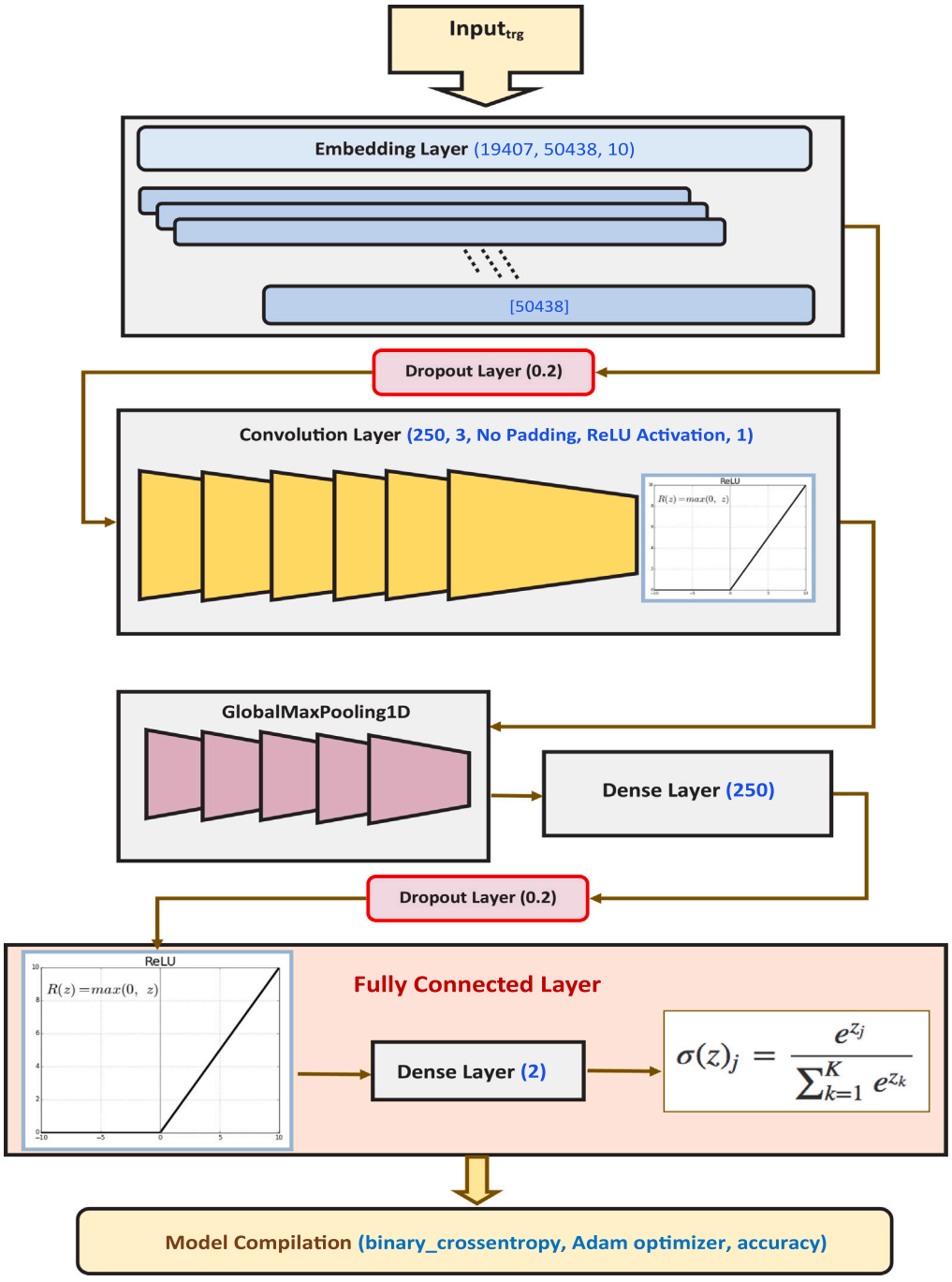

Fig. 11. 1D Convolutional Neural Network in detail.

3. $\boldsymbol{L}$ is fixed and represents the number of HTTP Request parameters being used as our feature set
4. Kernel size $\boldsymbol{w}$ defines the size of the sliding window to be convolved over layer input
5. Filters parameter $\boldsymbol{Z}$, that shows the total number of sliding windows of size $\boldsymbol{w}$
6. $\boldsymbol{R}$ representing the number of strides

The addition of embedding layer to our model, as shown in Eq. (1), map integers in the input dataset to embedding vectors having weights of $(\boldsymbol{W}, \boldsymbol{S})$.

$$\mathbb{M}_{\text{add}} = D(\mathbb{M}_{\text{add}}, \mathbb{E}(\boldsymbol{W}, \boldsymbol{S}, \boldsymbol{L})) \tag{1}$$

Since the output of Eq. (1) contains maximum of the total parameters on which convolution is going to take place, therefore, we want to avoid overfitting at this stage by adding a dropout layer to the model as shown in Eq. (2)

$$\mathbb{M}_{\text{add}} = D(\mathbb{M}_{\text{add}}, \mathbb{D}(r)) \ni r \ is \ the \ dropout \ rate \tag{2}$$

We now add the convolution layer to our CNN model with a kernel size $\boldsymbol{w}$ and filters $\boldsymbol{Z}$ along with the activation function and strides $\boldsymbol{R}$ as shown in Eq. (3).

$$\mathbb{M}_{\text{add}} = D(\mathbb{M}_{\text{add}}, \mathbb{C}onv(\boldsymbol{w}, \boldsymbol{Z}, \boldsymbol{A}_{ReLU}, \boldsymbol{R})) \tag{3}$$

At the next stage, Eq. (4) shows output $\boldsymbol{O_{im}}^{\boldsymbol{1}}$ of the first convolution layer. Here, bias for the mth feature map is represented as $\boldsymbol{B_{fm}}^{\boldsymbol{0}}$ and $\boldsymbol{K}$

represents the kernel weight.

$$O_{im}{}^{1} = A_{ReLU}\left(\sum_{x=1}^{w} K_{m,j}{}^{0}.f_{i+x}{}^{0} + B_{fm}{}^{0}\right) \tag{4}$$

Pooling layer is then added to our CNN model to reduce dimensionality, avoid over-fitting and only highlight the prominent features. We then add the fully connected layer to the model so that the layer has $\boldsymbol{n}$ neurons as shown in Eq. (5). We again add a dropout and activation layer as mentioned in Eqs. (2) and (3) respectively.

$$\mathbb{M}_{\text{add}} = D(\mathbb{M}_{\text{add}}, \mathbb{D}(n)) \tag{5}$$

We project our model onto an output layer that has two neurons for two classes by using the Softmax activation function, as shown in Fig. 11. This very combination of the fully connected layer and Softmax function are used for finding anomalies in the incoming HTTP request packets as it would classify the unknown test data into benign and malicious classes.

Finally, the model, with all its parameters is compiled and evaluated on both $\boldsymbol{D_{test}}$ and CSIC dataset (Giménez et al., 2012) to compute accuracy and binary cross-entropy loss for both the epochs.

3.5. Analyzing nested output of CAE and deep learning classifier

The outcomes of both Cookie Analysis engine and the Deep Learning based classifier are analyzed in order to decide about the nature of incoming HTTP request. The classifier's decision always supersedes the decision of CAE. One odd exception occurs when the, classifier categorizes a request as benign but the CAE considers it suspicious. Such a request is marked as suspicious as mentioned in Table 4 and the user is profiled accordingly. If a suspicious user again performs cookie mutation in the very next request, he is profiled malicious and the request is blocked without triggering the CAE or the classifier. This profiling of a user as suspicious is therefore helpful in both attacker profiling and reduction of processing time.

Table 4
Merging CAE with deep learning classifier.

CAE's Decision	–	–	B	B	M	M	S	S
DL Classifier's Decision	B	M	B	M	B	M	B	M
Final Decision	B	M	B	M	M	M	S	M

4. Framework analysis, results and discussion

There are certain metrics which help in theoretically analyzing the proposed framework before its implementation evaluation. Our proposed attack detection system would receive both malicious and benign HTTP requests which it needs to correctly determine with high accuracy. Let $\boldsymbol{R}$ be a set of requests and $\boldsymbol{M}$ and $\boldsymbol{B}$ are malicious and benign requests respectively. Let $\boldsymbol{M}_{\text{d}}$ and $\boldsymbol{B}_{\text{d}}$ be the correctly classified malicious and benign requests. The system will only yield high accuracy if Eqs. (6) and (7) are satisfied.

$$\boldsymbol{M}_{\text{d}} \approx \boldsymbol{M}. \tag{6}$$

$$\boldsymbol{B}_{\text{d}} \approx \boldsymbol{B}. \tag{7}$$

Moreover, optimizing the deep learning classifier by enhancing its performance was one of our key objectives. Suppose, a user $\boldsymbol{U}$ generates a total of $\boldsymbol{n}$ requests that includes both $\boldsymbol{M}$ and $\boldsymbol{B}$ queries at different time intervals $\boldsymbol{t_i}$. Suppose ∇ denotes the time deep learning classifier takes to correctly classify one incoming request. Now, with simple application of the classifier, for all $\boldsymbol{n}$ requests by the user, total execution time of the classifier ∇_{total} is mentioned in Eq. (8).

$$\nabla_{\text{total}} = \sum_{i=1}^{n} \nabla \tag{8}$$

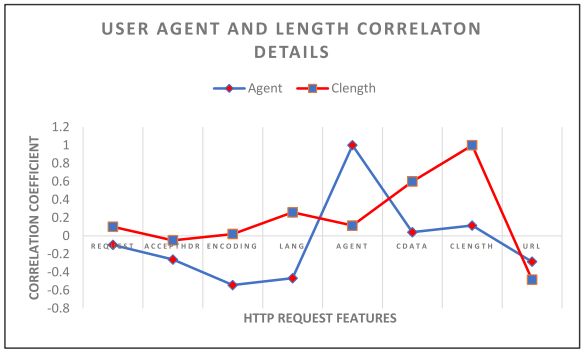

Fig. 12. Correlation details of C-length and user-agent.

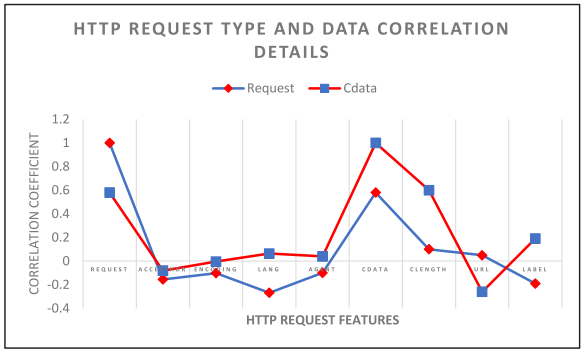

Fig. 13. Correlation details of request type and data.

Since we have optimized the classifier by introducing attacker profiling through the cookie analysis engine, therefore the classifier is not triggered once the attacker has been successfully profiled. Suppose the user $\boldsymbol{U}$ was sending benign requests but suddenly sent a malicious payload in his request $\boldsymbol{p_i}$, such that $1 \leq p \leq n$. This time, due to attacker profiling the optimized system results in lesser number of classifier executions as mentioned in Eq. (9). Thorough testing and evaluation under normal circumstances will help us give percentage decrease in the number of classifier executions which would further validate the proposed optimization.

$$\nabla_{\text{total}} = \sum_{i=1}^{n} \nabla - \sum_{i=p}^{n} \nabla \tag{9}$$

Based on the theoretical analysis, the proposed web attack detection, mitigation and attacker profiling framework was carefully analyzed and tested using a two-fold approach where we first tested our classifier on our own testing dataset and then on a publicly available benchmark dataset. We later tested our framework by deploying it in a live environment to analyze its performance and the benefits of attacker profiling. Moreover, the custom dataset we generated for both training and testing was also analyzed in order to validate our feature engineering.

Different HTTP parameters were found to actually correlate with others which reveals their interdependence. Correlation details of features *User agent* and *Content length* are shown in Fig. 12 while correlation details of features *Request Type* and *Data* are shown in Fig. 13.

4.1. Performance on our own dataset

The proposed deep learning classifier was first tested against $\boldsymbol{D_{test}}$. It successfully classifies almost all malicious and benign requests (*overall accuracy: 99.94%*) with very few exceptions where a large number of permutations are performed on different request fields. A comparison of loss and accuracy on $\boldsymbol{D_{test}}$ is shown in Fig. 14 while other performance parameters are shown in Fig. 15.

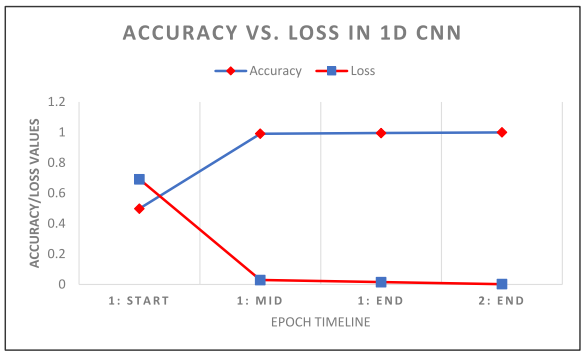

Fig. 14. Classifier accuracy vs. loss on D_{test}.

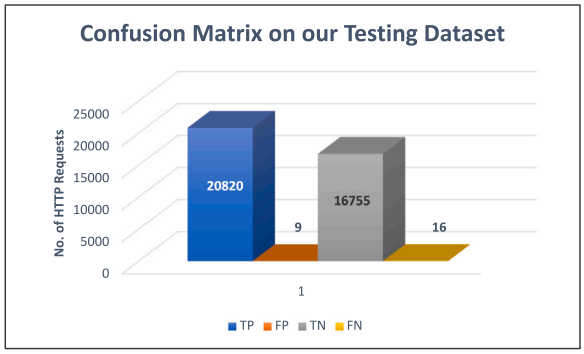

Fig. 15. Confusion matrix on test data.

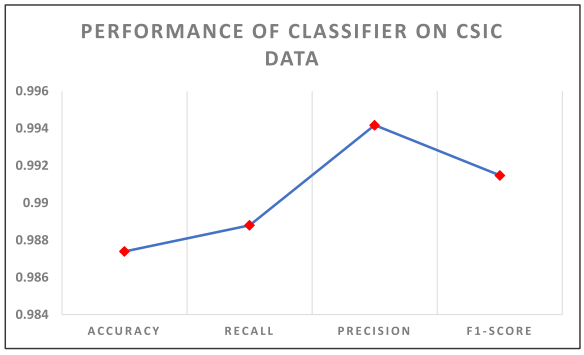

Fig. 16. Performance parameters on CSIC (Giménez et al., 2012) data.

4.2. Cross-dataset testing and validation

The proposed framework was further tested against a completely unknown publicly available benchmark CSIC dataset (Giménez et al., 2012). On this totally unknown data, a high accuracy would eliminate all chances of overfitting in the system. The proposed deep learning based classifier gives very high values of True Positive and True Negative as shown in Fig. 16 while accuracy, recall, precision and F-score on this public benchmark dataset are shown in Fig. 17.

Moreover, the minimum loss and maximum accuracy during training and testing (on CSIC Giménez et al., 2012) of proposed deep learning classifier is further highlighted in Figs. 18 and 19.

4.3. Discussion on cookies

While analyzing cookies in both datasets, it was observed that most of the malicious requests had normal benign cookie fields. Similarly, a few requests which were categorized as benign by the classifier had cookies which failed sanitization checks as shown in Fig. 20.

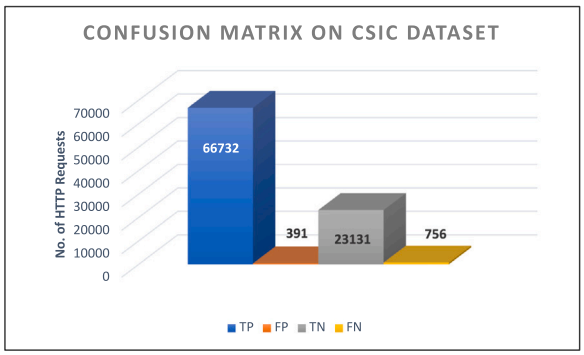

Fig. 17. Confusion matrix on CSIC (Giménez et al., 2012) data.

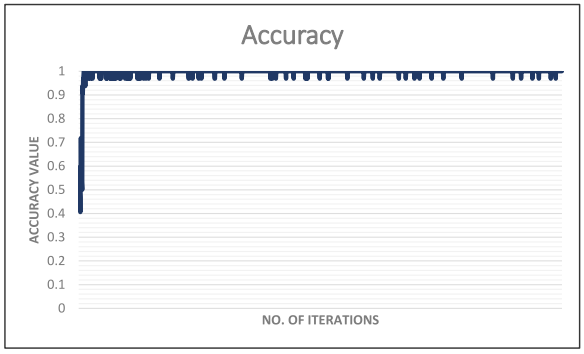

Fig. 18. CNN accuracy values on CSIC (Giménez et al., 2012) data.

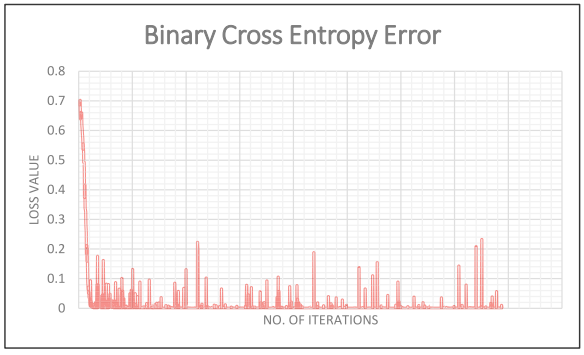

Fig. 19. CNN loss values on CSIC (Giménez et al., 2012) data.

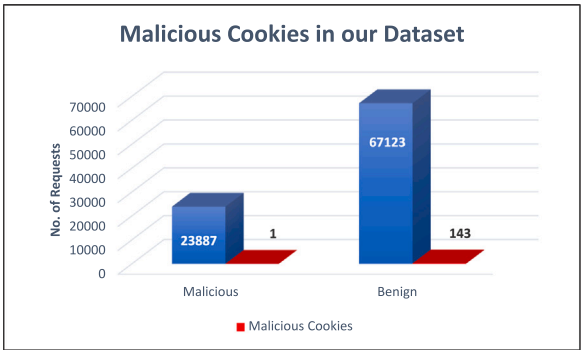

Fig. 20. Malicious cookies in both datasets.

This phenomena reflects that attackers prefer inserting their malicious scripts and payloads in other HTTP parameters. Moreover, the proposed Cookie Analysis Engine detected malicious cookies in publicly

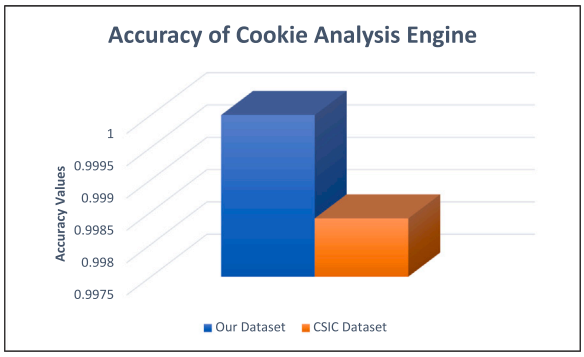

Fig. 21. Accuracy of CAE for both datasets.

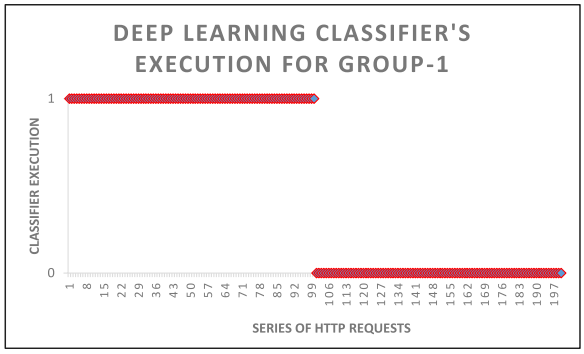

Fig. 22. Deep learning classifier execution for group-1.

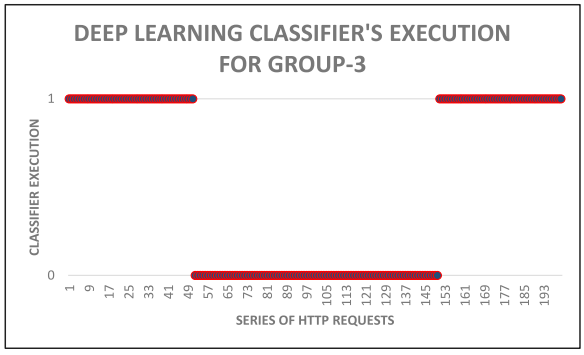

Fig. 23. Deep learning classifier execution for group-3.

available CSIC dataset with a very high accuracy, and in our own dataset with complete accuracy as shown in Fig. 21.

4.4. Deployment and testing of proposed framework in real time

The proposed framework has been designed to work efficiently in real time with high accuracy and minimized latency. As the CAE helps in maintaining state in case of a stateless HTTP protocol, testing the proposed framework against offline datasets would not offer complete benefits of the system. Nevertheless, evaluation of individual components (Deep Learning classifier and CAE) of the framework on these datasets in a silo has already hinted about very good performance of the proposed framework when deployed in real time.

4.4.1. Environment setup

We deployed the proposed framework in front of a sample website in a controlled environment. There are five groups of users $\boldsymbol{G}$=(G_1, G_2, G_3, G_4, G_5) who accessed the website under different scenarios by sending fifty requests each in a phase wise manner.

Initially G_1, G_2 and G_3 accessed the website normally while G_4 and G_5 used a scanner which were blocked by the bot and scanner detection engine. In the next phase, G_4 and G_5 accessed the site normally, G_1 and G_2 requested static pages, while G_3 contaminated their HTTP request by inserting a script in *data* field. In the next phase, G_2, G_4 and G_5 sent benign requests where the later requested a dynamic page, G_1 modified the cookie which still passed the sanitization checks and G_3 added third party cookies to their HTTP requests. In the last phase, G_1 again performed cookie mutation, cookies of G_5 failed sanitization checks while benign requests are sent by G_2, G_4 and G_3, with the later requesting a dynamic page. These request packets were generated to test and validate all modules of the proposed framework in a live environment. A Summary of User profiles at the end of each case is depicted in Table 5.

Table 5
Summarization of user profiling.

Phase	Benign Groups	Malicious Groups	Suspicious Groups
1	G_1, G_2, G_3	G_4, G_5	*0*
2	G_1, G_2, G_4, G_5	G_3	*0*
3	G_2, G_4, G_5	G_3	G_1
4	G_2, G_4, G_3	G_1	G_5

4.4.2. Comparison with existing literature

A major drawback of deep learning classifiers discussed in Section 1 is consumption of processing time which incurs latency in the system. Many research works mentioned in Section 2 employ deep learning without much concern about the processing time. This issue becomes more serious in case of real time systems where user behavior is changing rapidly. Our proposed framework has also made use of the deep

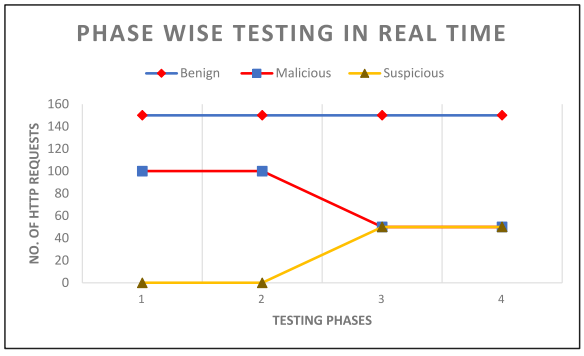

Fig. 24. Phase wise launching of different HTTP requests.

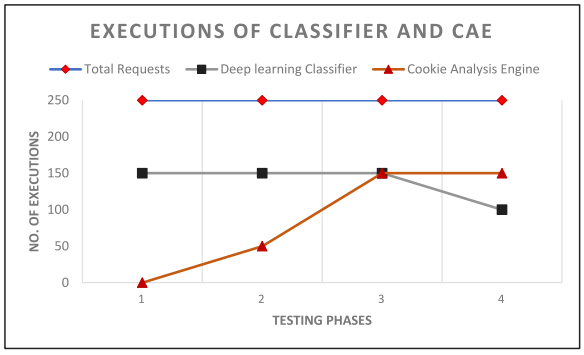

Fig. 25. Details of CNN classifier and CAE execution.

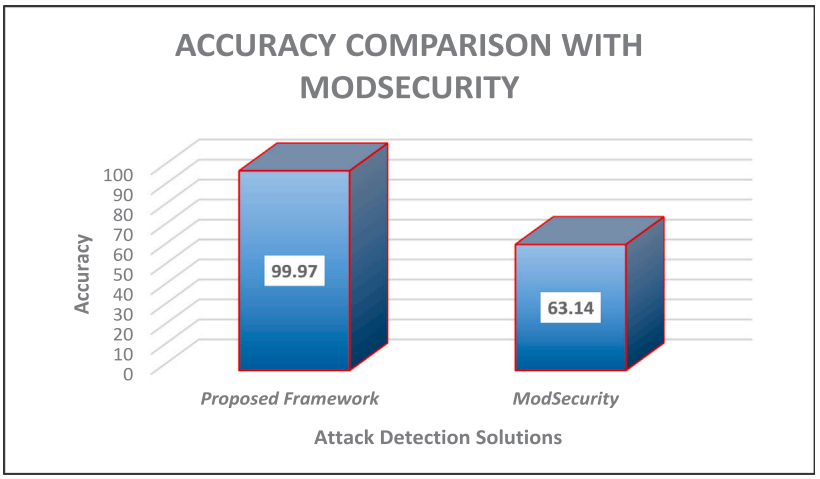

Fig. 26. Performance comparison: Proposed framework Vs. ModSecurity WAF.

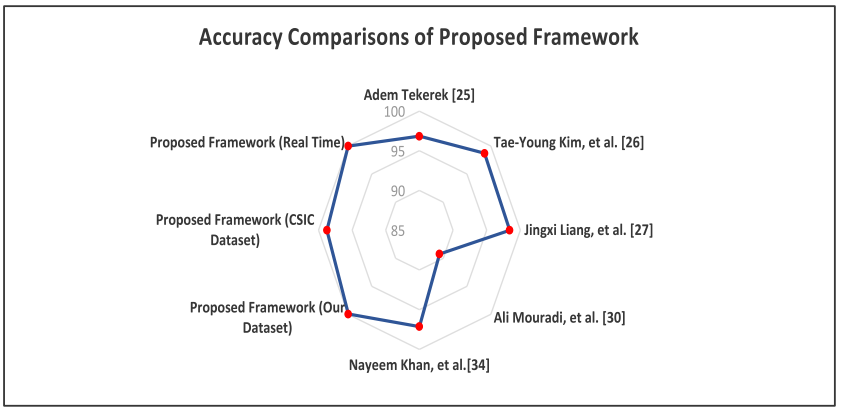

Fig. 27. 1D Convolutional Neural Network in detail.

learning based classifier but in an optimized way so that it does not get triggered in all cases as shown in Fig. 25, which saves both access time and processing power without any degradation in attack detection capability. In our phase wise testing, we executed, a total of *1000* requests as shown in Fig. 24. If we were to pass all incoming requests to the deep learning classifier, it would get executed every time. Looking individually at classifier execution for certain groups, we realized that by maintaining maximum possible accuracy, the classifier's execution dropped to 50% in the case of G_1 for all phases as mentioned in Fig. 22. Since G_3 turned malicious at *51st* request and duly profiled then, all later requests did not trigger the classifier resulting in optimal performance without hurting accuracy and precision. Executions for G_3 resume at *151st* request as it returns back to normalcy as shown in Fig. 23. In this case the proposed optimization enables the deep learning classifier to get triggered for only 51% of requests by G_3.

Due to the key feature of attacker profiling and cascading nature of our framework, our classifier was only triggered *550* times overall, while the CAE was triggered *350* times as displayed in Fig. 24 as shown in Fig. 25. The proposed framework yields an accuracy of 99.97% on this real time testing. We also exposed ModSecurity (Trustwave, 2020) to the same real world attack experiment and found out its dismal performance as compared to our proposed framework as shown in Fig. 26.

Fig. 27 provides comparative analysis in terms of accuracy of the proposed framework (for all three testing scenarios) with exiting approaches. It is pertinent to mention here that none of these approaches address the key issue of processing delays which has been duly addressed and catered for in our proposed framework.

Moreover, most approaches in Section 2 focus on a limited number of web attacks. For instance, Althubiti et al. (2017) only detects SQLi, XSS and object de-serialization, Jacob et al. (2012) focuses on URL, cookie and user-agent infection and Birkinshaw et al. (2019) only detects SQLi and XSS attacks, Pham et al. (2016), Recht (2019), Pan et al. (2019), Dewhurst (2020) just cater for XSS attacks. Moreover, Simos et al. (2019) focuses on port scanning and DDoS, Nguyen and Hwang (2016) only detects website defacement while Appelt et al. (2018), Seyyar et al. (2017), Yang et al. (2019), Zhou and Wang (2019), Stevanovic et al. (2011), The Apache Software Foundation (2020) detect the OWASP top ten attacks. In comparison, our proposed framework detects all major kinds of web attacks and saves useful time and processing power because of the attacker profiling feature.

5. Conclusion and future work

As websites carry huge amount of critical data, therefore the urge to maintain their security and privacy is of paramount importance. This research work proposes a framework where Deep Learning based classifier is nested with a Cookie Analysis Engine in order to protect a website from a wide variety of web attacks along with attacker profiling. For that purpose, a custom dataset was created for training while testing of the system was performed on a publicly available benchmark dataset completely unknown for the trained system. The proposed framework was also deployed and tested in a controlled real-time environment.

In our research we have demonstrated that the proposed Deep Learning classifier yields very high values of accuracy on all testing

datasets. The Cookie Analysis engine whose prime objective is to differentiate suspicious HTTP requests from the malicious ones, complements the functionality of our classifier by working in cohesion. The attacker profiling feature limits the execution of our classifier and cookie analysis engine, thereby saving processing resources in real time. The framework also caters for POST HTTP requests, unlike many other research works in the same domain. To summarize, we have presented a web attack detection, mitigation and attacker profiling framework, that detects web attacks with a very high accuracy and can be deployed in any scenario to safeguard a website. This paper also presents a detailed comparison of existing deep learning based techniques for detecting web attacks.

The main task we intend doing in near future is to develop a complete Web Deception system which inhibits key characteristics of the actual website, having a variety of deceptive lures. Later we will join our proposed framework with the deception system in such a way that anomalous requests are routed not to the actual website but the deception system where the attacker's engagement is enhanced in order to study and analyze the attack methodology, tactics and our weaknesses.

CRediT authorship contribution statement

Waleed Bin Shahid: Conceptualization, Methodology, Dataset generation, Development (Python, Machine Learning and Deep Learning based Classifers), Testing, Validation, Formal analysis, Writing and documentation . **Baber Aslam:** Conceptualization, Validation, Investigation of methodology and results, Writing – review & editing. **Haider Abbas:** Conceptualization, Validation, Formal analysis, Investigation of methodology and results, Writing and documentation, Writing – review & editing . **Saad Bin Khalid:** Methodology, Dataset generation . **Hammad Afzal:** Development (Python, Machine Learning and Deep Learning based Classifiers), Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research is sponsored by the Higher Education Commission (HEC), Pakistan through its initiative of National Center for Cyber Security for the affiliated lab National Cyber Security Auditing and Evaluation Lab (NCSAEL), Grant No: 2(1078)/HEC/ME/2018/707.

All authors have read and agreed to the published version of the final manuscript.

References

Abaimov, S., Bianchi, G., 2019. CODDLE: Code-injection detection with deep learning. IEEE Access 7, 128617–128627.

Althubiti, Sara A., Yuan, Xiaohong, Esterline, Albert C., 2017. Analyzing HTTP requests for web intrusion detection.

Appelt, D., Nguyen, C.D., Panichella, A., Briand, L.C., 2018. A machine-learning-driven evolutionary approach for testing web application firewalls. IEEE Trans. Reliab. 67 (3), 733–757.

Arachni, 2020. Web application security scanner framework. https://www.arachni-scanner.com/, [Online; accessed 26-March-2020].

Betarte, G., Pardo, Á., Martínez, R., 2018. Web application attacks detection using machine learning techniques. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 1065–1072.

Birkinshaw, Celyn, Rouka, Elpida, Vassilakis, Vassilios, 2019. Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks. J. Netw. Comput. Appl. 136, http://dx.doi.org/10.1016/j.jnca.2019.03.005.

Bitzer, Michael, Brinz, Nicolas, Ollig, Philipp, 2021. Disentangling the concept of information security properties-enabling effective information security governance.

Chawla, Nitesh V, Bowyer, Kevin W, Hall, Lawrence O, Kegelmeyer, W Philip, 2002. Smote: synthetic minority over-sampling technique. J. Artificial Intelligence Res. 16, 321–357.

Chris Sullo, David Lodge, 2020. Nikto. https://github.com/sullo/nikto, [Online; accessed 26-March-2020].

Clincy, V., Shahriar, H., 2018. Web application firewall: Network security models and configuration. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 01, pp. 835–836.

Cukier, Michel, 2007. Study: Hackers attack every 39 seconds. https://eng.umd.edu/news/story/study-hackers-attack-every-39-seconds, [Online; accessed 27-March-2020].

Dewhurst, Ryan, 2020. Damn vulnerable web application (DVWA). https://github.com/ethicalhack3r/DVWA, [Online; accessed 3-April-2020].

Dong, Ying, Zhang, Yuqing, 2017. Adaptively detecting malicious queries in web attacks.

Fang, Yong, Li, Yang, Liu, Liang, Huang, Cheng, 2018. Deepxss: Cross site scripting detection based on deep learning. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence. In: ICCAI 2018, Association for Computing Machinery, New York, NY, USA, pp. 47–51. http://dx.doi.org/10.1145/3194452.3194469.

Giménez, Carmen Torrano, Villegas, Alejandro Pérez, Marañón, Gonzalo Álvarez, 2012. Http dataset csic 2010. https://www.isi.csic.es/dataset/, [Online; accessed 1-April-2020].

Gomer, R., Rodrigues, E.M., Milic-Frayling, N., Schraefel, M.C., 2013. Network analysis of third party tracking: User exposure to tracking cookies through search. In: 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). 1, pp. 549–556. http://dx.doi.org/10.1109/WI-IAT.2013.77.

Hamam, Habib, Derhab, Abdelouahid, 2021. An owasp top ten driven survey on web application protection methods. In: Risks and Security of Internet and Systems: 15th International Conference, CRiSIS 2020, Paris, France, November 4-6, 2020, Revised Selected Papers. Vol. 12528, Springer Nature, p. 235.

Jacob, Grégoire, Kirda, Engin, Kruegel, Christopher, Vigna, Giovanni, 2012. Pubcrawl: protecting users and businesses from crawlers. p. 25.

Jemal, Ines, Haddar, Mohamed Amine, Cheikhrouhou, Omar, Mahfoudhi, Adel, 2021. ASCII embedding: an efficient deep learning method for web attacks detection. Pattern Recognit. Artif. Intell. 1322, 286.

Kandpal, Neeraj, Bandil, Devesh Kumar, Shekhawat, M.S., 2021. Improving website by analysis of web server logs using web mining tools. In: Advances in Information Communication Technology and Computing. Springer, pp. 525–531.

Khan, Nayeem, Abdullah, Johari, Khan, Adnan Shahid, 2017. Defending malicious script attacks using machine learning classifiers. Wirel. Commun. Mob. Comput. 2017, 5. http://dx.doi.org/10.1155/2017/5360472.

Kim, Tae-Young, Cho, Sung, 2018. Web traffic anomaly detection using C-LSTM neural networks. Expert Syst. Appl. 106, http://dx.doi.org/10.1016/j.eswa.2018.04.004.

Liang, Jingxi, Zhao, Wen, Ye, Wei, 2017. Anomaly-based web attack detection: A deep learning approach. In: Proceedings of the 2017 VI International Conference on Network, Communication and Computing. In: ICNCC 2017, Association for Computing Machinery, New York, NY, USA, pp. 80–85. http://dx.doi.org/10.1145/3171592.3171594, URL https://doi.org/10.1145/3171592.3171594.

Liedke, Lindsay, 2020. 100+ Internet statistics and facts for 2020. https://www.websitehostingrating.com/internet-statistics-facts/, [Online; accessed 29-March-2020].

Liu, Tianlong, Qi, Yu, Shi, Liang, Yan, Jianan, 2019. Locate-then-detect: Real-time web attack detection via attention-based deep neural networks. pp. 4725–4731. http://dx.doi.org/10.24963/ijcai.2019/656.

Luo, C., Tan, Z., Min, G., Gan, J., Shi, W., Tian, Z., 2020. A novel web attack detection system for internet of things via ensemble classification. IEEE Trans. Ind. Inf. 1. http://dx.doi.org/10.1109/TII.2020.3038761.

Mahfoudhi, Adel, 2021. Malicious http request detection using code-level convolutional neural network. In: Risks and Security of Internet and Systems: 15th International Conference, CRiSIS 2020, Paris, France, November 4-6, 2020, Revised Selected Papers. Vol. 12528, Springer Nature, p. 317.

Max Roser, Hannah Ritchie, Ortiz-Ospina, Esteban, 2020. Internet. Our World Data https://ourworldindata.org/internet.

Meng, N., Nagy, S., Yao, D., Zhuang, W., Arango-Argoty, G., 2018. Secure coding practices in java: Challenges and vulnerabilities. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 372–383.

Mokbal, F.M.M., Dan, W., Imran, A., Jiuchuan, L., Akhtar, F., Xiaoxi, W., 2019. MLPXSS: An integrated XSS-based attack detection scheme in web applications using multilayer perceptron technique. IEEE Access 7, 100567–100580.

Moradi Vartouni, A., Teshnehlab, M., Sedighian Kashi, S., 2019. Leveraging deep neural networks for anomaly-based web application firewall. IET Inf. Secur. 13 (4), 352–361.

Nair, Sumit, 2019. Web application firewall (WAF) solutions. https://mobisoftinfotech.com/resources/wp-content/uploads/2018/05/AWS-WAF-Banner.png, [Online; accessed 21-March-2020].

Nguyen, T. K., Hwang, S. O., 2016. Large-scale detection of dom-based xss based on publisher and subscriber model. In: 2016 International Conference on Computational Science and Computational Intelligence (CSCI), pp. 975–980.

Niu, Q., Li, X., 2020. A high-performance web attack detection method based on CNN-gru model. In: 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). Vol. 1, pp. 804–808. http://dx.doi.org/10.1109/ITNEC48623.2020.9085028.

OWASP, 2020. OWASP Zed attack proxy (ZAP). https://owasp.org/www-project-zap/, [Online; accessed 26-March-2020].

Paine, Laura, 2019. 2019 Verizon DBIR shows web applications and human error as top sources of breach. https://www.veracode.com/blog/security-news/2019-verizon-dbir-shows-web-applications-and-human-error-top-sources-breach, [Online; accessed 27-March-2020].

Pan, Yao, Sun, Fangzhou, Teng, Zhongwei, White, Jules, Schmidt, Douglas, Staples, Jacob, Krause, Lee, 2019. Detecting web attacks with end-to-end deep learning. J. Internet Serv. Appl. 10, http://dx.doi.org/10.1186/s13174-019-0115-x.

Pham, T. S., Hoang, T. H., Vu, V. C., 2016. Machine learning techniques for web intrusion detection — A comparison. In: 2016 Eighth International Conference on Knowledge and Systems Engineering (KSE), pp. 291–297.

Portswigger Web Security, 2020. Burpsuite. https://portswigger.net/burp, [Online; accessed 28-March-2020].

Recht, Benjamin, 2019. A tour of reinforcement learning: The view from continuous control. ArXiv http://arxiv.org/abs/1806.09460.

Sarkar, Sandip, 2021. Detecting vulnerabilities of web application using penetration testing and prevent using threat modeling. In: Advances in Electronics, Communication and Computing. Springer, pp. 21–32.

Seyyar, Merve, Catak, Ferhat Ozgur, Gul, Ensar, Merve, B.S., 2017. Detection of attack-targeted scans from the apache HTTP server access logs. Appl. Comput. Inf. 14, http://dx.doi.org/10.1016/j.aci.2017.04.002.

Shahid, Waleed Bin, Abbas, Haider, Aslam, Baber, Afzal, Hammad, Khalid, Saad Bin, 2021. A framework to optimize deep learning based web attack detection using attacker categorization. [Accepted in IEEE Wetice 2021].

Shoel, H., Jaatun, M. G., Boyd, C., 2018. Owasp top 10 - do startups care? In: 2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), pp. 1–8.

Simos, D. E., Zivanovic, J., Leithner, M., 2019. Automated combinatorial testing for detecting sql vulnerabilities in web applications. In: 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), pp. 55–61.

Stevanovic, Dusan, An, Aijun, Vlajic, Natalija, 2011. Detecting web crawlers from web server access logs with data mining classifiers. In: Kryszkiewicz, Marzena, Rybinski, Henryk, Skowron, Andrzej, Raś, Zbigniew W. (Eds.), Foundations of Intelligent Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 483–489.

Tekerek, Adem, 2021. A novel architecture for web-based attack detection using convolutional neural network. Comput. Secur. 100, 102096. http://dx.doi.org/10.1016/j.cose.2020.102096, URL http://www.sciencedirect.com/science/article/pii/S0167404820303692.

The Apache Software Foundation, 2020. Apache module mod_dumpio. https://httpd.apache.org/docs/2.4/mod/mod_dumpio.html, [Online; accessed 3-April-2020].

The Open Web Application Security Project, 2017. OWASP top ten. https://owasp.org/www-project-top-ten/, [Online; accessed 25-March-2020].

Tian, Z., Luo, C., Qiu, J., Du, X., Guizani, M., 2020. A distributed deep learning system for web attack detection on edge devices. IEEE Trans. Ind. Inf. 16 (3), 1963–1971.

Trustwave, 2020. ModSecurity Open source web application firewall. https://modsecurity.org/, [Online; accessed 28-March-2020].

Vicente, Gorka, 2019. The top 5 reasons why WAF users are dissatisfied. https://hdivsecurity.com/bornsecure/the-top-5-reasons-why-waf-users-are-dissatisfied/, [Online; accessed 22-March-2020].

Wen, Yujun, Yuan, Hui, Zhang, Pengzhou, 2016. Research on keyword extraction based on word2vec weighted textrank. In 2016 2nd IEEE International Conference on Computer and Communications (ICCC), pp. 2109–2113.

Wittern, E., Ying, A. T. T., Zheng, Y., Dolby, J., Laredo, J. A., 2017. Statically checking web api requests in javascript. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp. 244–254.

Yang, Jing, Wang, Liming, Xu, Zhen, Wang, Jigang, Tian, Tian, 2019. Coordinated web scan detection based on hierarchical correlation. In: Li, Jin, Liu, Zheli, Peng, Hao (Eds.), Security and Privacy in New Computing Environments. Springer International Publishing, Cham, pp. 388–400.

Zhang, Jingchi, Jou, Yu-Tsern, Li, Xiangyang, 2019. Cross-site scripting (XSS) detection integrating evidences in multiple stages. In: HICSS.

Zhang, Ming, Xu, Boyi, Bai, Shuai, Lu, Shuaibing, Lin, Zhechao, 2017. A deep learning method to detect web attacks using a specially designed CNN. In: Liu, Derong, Xie, Shengli, Li, Yuanqing, Zhao, Dongbin, El-Alfy, El-Sayed M. (Eds.), Neural Information Processing. Springer International Publishing, Cham, pp. 828–836.

Zhou, Yun, Wang, Peichao, 2019. An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. Comput. Secur. 82, 261–269. http://dx.doi.org/10.1016/j.cose.2018.12.016, URL http://www.sciencedirect.com/science/article/pii/S0167404818306370.

Waleed Bin Shahid is an Assistant Professor at the Department of Information Security at National University of Sciences and Technology (NUST), Islamabad, Pakistan. He has over a decade of practical and research experience in different domains of Cyber Security and has certain IF publications to his credit, including one in JNCA in 2018. His areas of research interests are Web Security, Deception systems, Malware Analysis and Digital Forensics.

Dr. Baber Aslam is working as an adjunct Associate Professor at Department of Information Security at National University of Sciences and Technology (NUST), Islamabad, Pakistan. He holds doctoral degree in cyber security from University of Central Florida, USA. He has also served as head of the information security department at NUST. Dr. Baber has 5 IF publications and 15 conference papers to his credit. His research interests include Digital Forensics, Honeypots and Network Security.

Dr. Haider Abbas is Senior Member IEEE and a Cyber Security Professional who took professional trainings and certifications from Massachusetts Institute of Technology (MIT) United States, Stockholm University Sweden, IBM and EC-Council. He received his M.S. in Engineering and Management of Information Systems (2006) and Ph.D. in Information Security (2010) from KTH-Royal Institute of Technology, Sweden. Dr. Abbas has received several research grants for ICT related projects from various research funding authorities and has been working on scientific projects in US, EU, KSA and Pakistan. His professional services include – but are not limited to – Journal Editorships, Industry Consultations, Workshops Chair, Technical Program Committee Member, Invited/Keynote Speaker and reviewer for several international journals and conferences. He has authored over 150 scientific research articles in prestigious international journals and conferences.

Saad Bin Khalid is a Research Assistant at National Cyber Security Auditing and Evaluation Lab which is affiliated with the Department of Information Security at National University of Sciences and Technology (NUST), Islamabad, Pakistan. His research interests include vulnerability exploitation and defense, web security and web development.

Dr. Hammad Afzal is an Associate Professor at the Department of Computer Science at National University of Sciences and Technology (NUST), Islamabad, Pakistan. Dr. Hammad's research interests include Machine Learning, Artificial Intelligence and their applications in Cyber Security, medicine and image processing. He has over 20 IF publications over 15 conference papers to his credit.