Automated Classification of Web-Application Attacks for Intrusion Detection

Harsh Bhagwani, Rohit Negi, Aneet Kumar Dutta, Anand Handa(✉), Nitesh Kumar, and Sandeep Kumar Shukla

C3I Center, Department of CSE, Indian Institute of Technology, Kanpur, India
harshbhagwani@ymail.com,
{rohit,aneet,ahanda,niteshkr,sandeeps}@cse.iitk.ac.in

Abstract. In today's information driven society and economy, web facing applications are most common way to run information dissemination, banking, e-commerce etc. Web applications are frequently targeted by attackers through intelligently crafted http requests to exploit vulnerabilities existing in the application, front-end, and the web-clients. Some of the most frequent such attacks are SQL Injection, Cross-Site Scripting, Path-traversal, Command Injection, Cross-site request forgery etc. Detecting these attacks up front and blocking them, or redirecting the request to a honey-pot could be a way to prevent web applications from being exploited. In this work, we developed a number of machine learning models for detecting and classifying http requests into normal, and various types of attacks. Currently, the models are applied as an ensemble on the http server logs, to classify and build data analytics on the http requests received by any web server in order to garner threat intelligence, and threat landscape. We also implemented an online log-analysis version that analyzes logs every 15 s to classify http requests in the recent 15 s. However, it can also be used as a web application firewall to block the http requests based on the classification results. We also have implemented an intrusion protection mechanism by redirecting http requests classified upfront as malicious towards a web honeypot. We compare various existing signature based, regular expression based, and machine learning based techniques against our models for detection and classification of http based attacks, and show that our methods achieve better performance over existing techniques.

Keywords: Intrusion detection system · Web security · Machine learning

1 Introduction

In the last few years, there has been a tremendous growth in the use of the Internet. According to the Internet World statistics, the number of Internet users has increased by 30% within a decade [10]. From booking an appointment for health check-ups to online purchasing, the Internet is becoming part and parcel of our lives. With the introduction of these web facing services, there has been a huge growth in the web-application attacks which necessitates timely detection of attacks and prevention. Malicious users are exploiting the vulnerabilities of these services to obtain the personal data of other users, and affecting the quality of service. According to the survey done by Sophos, India is ranked third in the world, in the number of cyber attacks in 2018 and around 76% of the Indian organizations faced cyber attacks in 2018 [9]. In another survey done by Small Business Trends, hackers have targeted 43% of the start-ups, and around 60% of them have gone out of business within a few months of the attacks [24].

According to the ENISA Threat Landscape report (ETL) of 2018 [18], web application attack is ranked 3rd, consecutively for two years, among other attacks in the cyber domain. Among the various types of web application attacks, there are different techniques of different levels of difficulties, and different damages they cause. In ETL 2018 report, SQL injection [5] dominates the web application attacks with a share of 51%. Local File Inclusion and Cross-Site Scripting (XSS) [7] come 2nd and 3rd in the attack ranking respectively. Attacks like injection (mainly including SQL, Lightweight Directory Access Protocol (LDAP) [29], XML Path (XPath) [2], etc.) and XSS come under one of the categories in the OWASP Top 10 list of the most common attacks [28]. According to a report published by Positive Technologies in 2017 [35], SQL injection and Cross-site Scripting together have contributed about 50% of the attacks.

Intrusion Detection Systems (IDS) are security applications that monitor a computer network or hosts therein, to detect any malicious activities or threats. On detection, they alert the system administrators [17,23,32]. Alerting may vary from logging the attacks in log files or alerting an administrator on security dashboards to take necessary actions. Ideally, an Intrusion Detection System should be simple, fast and precise. But it is impossible for an IDS to provide complete protection as no detection method is 100% accurate. They make two types of errors: False Positive - this is generally when a normal access is falsely considered as a threat and False Negative - when a real attack goes undetected by the IDS systems. Although, false positives are acceptable because they are passed to the administrator for reviews, too many false positives can be a burden for them as well. However, false negatives are never reviewed by the administrator as they are mistaken as non-malicious threats. Hence, most of the organizations tune their IDS from time to time to ensure close to zero false negatives, and close to tolerance level of false positives.

One could use many different techniques to implement an IDS. Signature based techniques use existing database of patterns for malicious access which are matched against incoming activities (in web-application case, http requests). Regular expression based signature schemes generalize the static patterns for such detection. However, in this work, we show that these methods are often not very effective in reducing false negatives or false positives. Also, as attackers apply more intelligent encoding schemes to hide their malicious payload, static or regular expression based patterns can be easily bypassed. Rule based techniques are another generalization of pattern matching. Dynamic learning of rules for attack detection with reinforcement learning could be one way to mitigate these problems – however, that requires complex training process, and requires administrators to be interactive. Customers of IDS often want out-of-the-box solutions rather than those requiring training on-site. Machine learning based on past attack payloads therefore have been proposed by a number of researchers. However, most prior work seem to be focused on detection of SQL injection, and Cross-site scripting attacks. Our goal in this work has been to build a tool that uses an ensemble of models trained to detect a larger variety of attacks, and use the ensemble to classify the http requests into different attack types. We also target offline detection from log files, and online detection from incremental log files of http servers. Further, we are in the process of applying the same ensemble of models to detect attacks even before the request has been executed by the web application – thereby creating a web-application firewall. However, in this paper, we only discuss the log file based detection. One might question the need for classification/detection after the request has been processed – by analyzing after the fact, from log files. This is useful when one has a web application that is highly secure with all vulnerabilities patched, hence the malicious requests have no effect even when processed, but after the fact analysis provides system administrators with statistics of various kinds of attack attempts, the payload structures, and thereby the threat landscape.

Therefore, the models presented in this paper feeds on the log files of the web applications to detect attacks like SQLi, XSS, path traversal [11,26], OS command [30], Server Side Includes (SSI) [34], LDAP, CRLF [6], and XPath. It is able to detect the mentioned eight most popular attacks along with any other anomalies which cannot be classified in one of the above classes of attacks. The same machine learning based tool can be turned into a firewall by intercepting http requests before it is delivered to the web application.

It is to be noted that a web server provides two different log files - access log and error log. The access log maintains the history of all the requests a web server receives and the error log maintains the records of all the error a web server encountered while processing the requests. This work focuses more on ‘access.log’ because it contains all the necessary information required to monitor the server. In Table 1 the various fields in a typical log file entry are shown.

Due to lack of space, we do not provide the definitions of the eight different web application attacks considered in this paper. Readers unfamiliar with the definitions of these attacks are referred to [2,6,11,12,29,30,34,36].

The rest of the paper is organized as: In Sect. 2, we discuss the existing regular expression based approaches and machine learning based approaches to detect or classify web application attacks. Section 3 discusses our proposed framework. In Sect. 4, we evaluate our proposed framework and discuss the results. Section 5 compares our framework with the existing approaches, and finally, we conclude the paper in Sect. 6 with some discussion on future work.

Table 1. Definition of different log fields

Fields	Definition
122.172.41.188	Client's IP address
–	The identity of the client on client's machine
–	Username of the client if the request was authenticated
$[24/Jun/2018{:}08{:}12{:}12+0000]$	Time at which the request was received
"GET /main.php HTTP/1.1"	This contains HTTP method, path of the requested resource and HTTP protocol version used by client
200	It is a status code send by the server to the client after processing the request
203	Size of the resources requested by client
–	Referrer - Indicates from where did the request arrived
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) Gecko/ 20100101 Firefox/45.0"	Gives information about the user making requests like web browser, OS used, website source etc

2 Related Work

In this section we outline the different existing approaches to detect or classify web application attacks.

2.1 Regular Expression Based Approaches

Regular Expression or Regex is a set of strings represented in an algebraic format which is used to describe the search pattern of a string [26]. This makes regex fit for text processing, data validation and searching of strings. Searching strings using regular expression is fast. A proposed method of detection of web application attacks from entries in http server log files using regular expressions, can be found in [26]. The paper proposes regular expressions to match http requests formed by attackers in cases of XSS, code injection attacks (such as SQLi, LDAPi, XPath and OS Command) and Path Traversal attack. The regular expressions may be able to detect both obfuscated and non-obfuscated version of attacks. They search for keywords like scripts, update, insert,../, etc. (which are found in these attacks) in the received requests. Although they do not provide any analysis of accuracy or precision of applying these regular expression on log files, still their prescribed regular expressions are widely used in many web application IDS. Table 2 shows the attack types and their corresponding regular expressions provided in [26] for SQLi, XSS, and Path Traversal. The regular expressions for other attacks are missing in [26]. Later in this paper, we measure the accuracy of detection using these regular expressions on test data to check their accuracy.

Table 2. Regular expressions used in [26].

Attack type	Regular expression
XSS	$1./((\backslash\%3C)\mid<)(\backslash\%2F)\mid\backslash/)*[az09\backslash\%]+((\backslash\%3E)\mid>)/ix$ $2./((\backslash\%3C)\mid<)((\backslash\%69)\mid i\mid(\backslash\%49))((\backslash\%6D)\mid m\mid(\backslash\%4D))$ $((\backslash\%67)\mid g\mid(\backslash\%47))[\wedge\backslash n]+((\backslash\%3E)\mid>)/I$ $3.\backslash(javascript\mid vbscript\mid expression\mid applet\mid script\mid embed\mid$ $object\mid iframe\mid frame\mid frameset)/i$
SQLi	$1./(\backslash')\mid(\backslash\%27)\mid(\backslash\backslash)\mid(\#)\mid(\backslash\%23)/ix$ $2./((\backslash\%3D)\mid(=))[\wedge\backslash n]*((\backslash\%27)\mid(\backslash')\mid(\backslash\backslash)\mid(\backslash\%3B)\mid(;))/i$ $3./\backslash w*((\backslash\%27)\mid(\backslash'))(\backslash s\mid\backslash+\mid\backslash\%20)*((\backslash\%6F)\mid o\mid$ $(\backslash\%4F))((\backslash\%72)\mid r\mid(\backslash\%52))/ix$ $4./((\backslash\%27)\mid(\backslash'))(select\mid union\mid insert\mid update\mid delete\mid$ $replace\mid truncate)/ix$ $5./exec(\backslash s\mid\backslash+)+(s\mid x)p\backslash w+/ix$
Path Traversal	$/(\backslash.\mid(\%\mid\%25)2E)(\backslash.\mid(\%\mid\%25)2E)(\backslash/(\%\mid\%25)2F\mid\backslash\backslash\mid(\%\mid$ $\%25)5C)/i$
SSI	Not Provided
OS Command	Not Provided
XPath	Not Provided
CRLF	Not Provided
LDAP	Not Provided

Table 3. Regular expressions used by [27]

Attack type	Regular expressions used by OWASP ModSecurity
XSS	$http:\backslash/\backslash/[\backslash w\backslash.]+?\backslash/.*?\backslash.pdf\backslash b[\wedge\backslash x0d\backslash x0a]*\#$
SQLi	No Regular Expression provided
Path Traversal	No Regular Expression provided
SSI	$<!--\backslash W*?\#\backslash W*?(?:e(?:cho\mid xec)\mid printenv\mid include\mid$ $cmd)$
OS Command	$(?i:(?:[\backslash;\backslash\mid\backslash`]\backslash W*?\backslash bcc\mid\backslash b(wget\mid curl))\backslash b\mid\backslash/(?:[\backslash'\backslash''\backslash\mid$ $\backslash;\backslash`\backslash-\backslash s]\mid\$))$
XPath	No Regular Expression provided
CRLF	No Regular Expression provided
LDAP	$(?:\backslash((?:W*?(?:objectc(?:ategory\mid lass)\mid homedirectory\mid$ $[gu]idnumber\mid cn)\backslash$ $b\backslash W*?=\mid[\wedge\backslash w\backslash x80-\backslash xFF]*?[\backslash!\backslash\&\backslash\mid$ $][\wedge\backslash w\backslash x80-\backslash xFF]*?\backslash()\backslash)[\wedge\backslash w\backslash x80-\backslash xFF]$ $*?\backslash[\wedge\backslash w\backslash x80-\backslash xFF]*?[\backslash!\backslash\&\backslash\mid])$

OWASP ModSecurity Core Rule Set (CRS)[27] gives the regular expressions to most of the attacks. They use the rule sets in ModSecurity, a firewall used to provide securities from web app attacks. Table 3 depicts the types of attacks and their corresponding regular expressions used to detect the attacks. In this work, we also measure the accuracy obtainable by these regular expression based matching in detecting different attack types. As we find in Sect. 4, we obtain better results in the detection of web application attacks using machine learning based approaches than regex based ones.

Now, we discuss some of the most well known approaches in web attack detection using the concept of machine learning.

2.2 Machine Learning Based Approaches

In [13], authors propose a machine learning method to detect anomalous web traffic. They perform their experiments on CSIC 2010 HTTP dataset. The dataset consists of 25065 anomalous requests and 36000 normal requests. In CSIC 2010 dataset, the anomalous request is a collection of various attacks like XSS, SQLi, CSRF, etc. Using Weka analysis, they obtain five best features from a set of 9 features based on their relevance and their impact on accuracy. The best five features are request length, arguments length, number of arguments, path length and number of special characters in the path. For classification of requests into normal and anomalous, they have used various machine learning techniques like Random Forest, Logistic Regression, Decision Tree, AdaBoost Classifier, Stochastic Gradient Descent Classifier, and Naive Bayes. With all the mentioned classifiers, they have achieved an accuracy of 99.94% except with Stochastic Gradient Descent for which they obtain 99.88% and with Naive Bayes they obtain 88.83%. Although they got good accuracy, they do not further classify the attacks into individual categories. We implement their model and applied to our dataset. The highest accuracy using Random Forest classifier came out to be 88.84%. This indicates that the model might not generalize well beyond the CSIC 2010 dataset.

In [25], the authors discuss the detection of Cross-site Scripting attacks using machine learning techniques. They dealt with both normal and obfuscation version of XSS attacks. For training, they collected 4000 requests from many different sites consisting of 2000 benign samples from Dmoz and ClueWeb09 and 2000 XSS attack samples from [21]. Their testing dataset includes 13000 each of benign and XSS samples. They prepare the data by removing duplicates, removing extra new lines and blank spaces and at last, changing all the characters of the request to lowercase. To get a good result, they select two types of features namely structural features and behavioral features. The structural features deals with punctuation like &, %, /, \, +, @, *space*, |, # and combinations of different punctuation like ><,′" ><, [], ==, &#. Behavioural features contain a list of selected commands and functions like `eval(), Onload, Onerror, createelement, String.fromCharCode, Search` which are usually found in an XSS attack. Together, they form a total of 38 structural features and 21 behavioral features. These features have been used to create binary feature vectors indicating the presence or absence of individual features in a request. To classify the attacks into XSS and benign samples, they used four classifiers: Random Forest, K-Nearest Neighbor and two variations of Support Vector Machine (Linear Kernel and Polynomial Kernel). After tuning the classifiers using 5-fold cross-validation, they tested the classifiers with test data and achieve an accuracy of 99.5% with Random Forest, 99.75% with K-NN, 99.6% with Polynomial Kernel SVM and 96.32% with Linear Kernel SVM. When we implement machine learning models with their feature set, we get the best accuracy of 89.81% with Random Forest.

Cheon et al. [20] propose a method to prevent SQL injection in web applications. They detect the SQLi attacks using Bayesian Classifier, a machine learning classifier using only two features which are the length of the parameters and the number of keywords of parameters. The keywords are the commands and symbols found in SQL statements like commas, quotation marks, "UNION," "SELECT", etc., but they do not specify the full set of keywords used in second feature set. They form their training dataset of size 2142 containing a mix of both SQL injection patterns and benign samples by using a python script and a test dataset of 4070 different patterns. After training the Bayesian Classifier with full training dataset, upon testing they achieve an accuracy of 99.61%. Also, they detect different patterns of SQL injection attacks. We are not able to reproduce their model due to lack of enough details in [20].

These most recent machine learning based approaches supersede a few other previous approaches, and hence not discussed here. In summary, our observations on the past approaches are as follows:

– We found no substantial amount of work to detect attacks like XPath injection, CRLF, LDAP Injection, OS Command, SSI attack and Path traversal attack using machine learning techniques so far.
– Only after the fact analysis is done in the detection of attacks on web application from http log files.
– Although, the above mentioned approaches have achieved high accuracy, but none of them has implemented the sub-classification of attacks.

3 Proposed Framework

In this section, we explain the architecture of our proposed log-based attack detection and classification methods to achieve better detection accuracy. We also discuss the implementation of our solution for a live website. We divide this section into two phases offline log based detection and classification, and online implementation.

3.1 Offline Log Based Detection and Classification

This step include dataset collection, data processing, feature extraction and modeling of processed data as shown in Fig. 1.

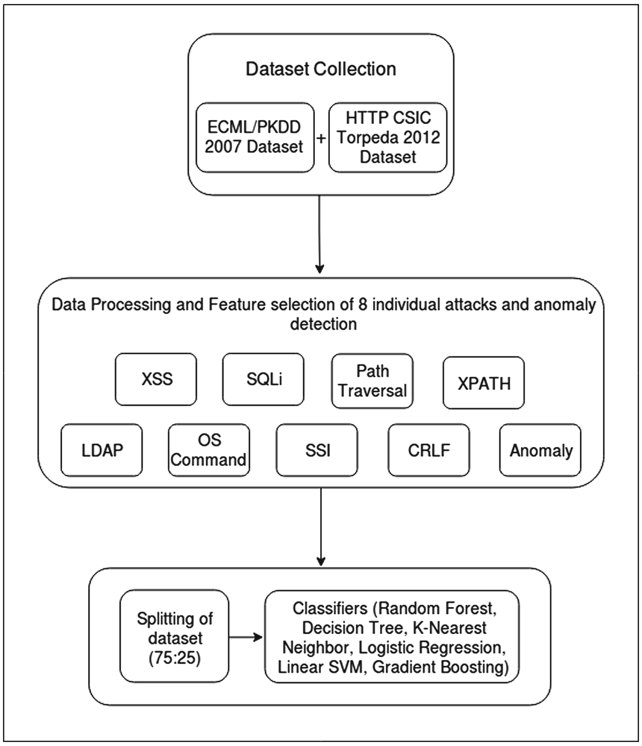

Dataset Collection
ECML/PKDD 2007 Dataset
+
HTTP CSIC Torpeda 2012 Dataset
Data Processing and Feature selection of 8 individual attacks and anomaly detection
XSS
SQLi
Path Traversal
XPATH
LDAP
OS Command
SSI
CRLF
Anomaly
Splitting of dataset (75:25)
Classifiers (Random Forest, Decision Tree, K-Nearest Neighbor, Logistic Regression, Linear SVM, Gradient Boosting)

Fig. 1. Architecture for Phase 1

Dataset Collection. We use a combination of two different datasets ECML/PKDD 2007 Discovery Challenge dataset [1] and HTTP CSIC Torpeda 2012 dataset [15].

1. **ECML/PKDD 2007 Dataset**
   ECML/PKDD 2007 Discovery Challenge aims to address the classification of malicious HTTP requests received by a web application and identification of the attack requests. They provide the real-time traffic of HTTP requests in XML format stored in .txt file. The dataset consists of a total of 50116 samples, out of which 35006 are valid requests and 15110 are different types of attack request.
2. **HTTP CSIC Torpeda 2012 Dataset**
   This dataset is created using Torpeda framework [16]. Torpeda framework is used to develop labeled web traffic. The motive of Torpeda is to generate a dataset of HTTP requests for the purpose of evaluating the effectiveness of a Web Application Firewall. The torpeda dataset is given in XML document.

The statistics of the final dataset used for our work is shown in Table 4. In an HTTP request, there are many fields like URL, Query, Protocol, Method, Path,

Table 4. Dataset used for detection of each attack.

Attack types	Attack requests	Bengin requests	Total requests	Training set	Testing set
XSS	5310	5138	10448	7836	2612
SQL	4418	4801	9219	6914	2305
Path Traversal	1933	2000	3933	2949	984
OS Command	2094	2200	4294	3220	1074
CRLF	319	337	656	492	164
SSI	1889	1999	3888	2916	972
LDAP	1791	1945	3736	2802	934
XPath	1913	2035	3948	2961	987
Anomaly	22258	28033	50291	37718	12573

etc. In this work, we focus only on Path and Query part of the HTTP request because payload of the attacks are encoded in these parts [22,37]. Figure 2 explains both the Path and Query parts of the HTTP requests.

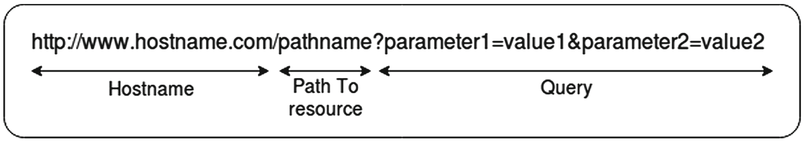

http://www.hostname.com/pathname?parameter1=value1&parameter2=value2
Hostname
Path To resource
Query

Fig. 2. Structure of an HTTP request

Data Processing. This section explains the data processing required to extract the necessary features. Data processing involves the following steps:

- URL decoding of the requests
- Converting the request to lowercase (except CRLF where we change the request to uppercase)
- Replacement of characters

These steps are almost same for all the attacks. The only variation is in the number of decoding of URLs and characters to be replaced for each attack. After data processing, we pass the processed data for feature extraction. Table 5 shows operations involved in data processing.

Feature Engineering. A feature is a measurable attribute based on which machine learning classifiers predict the results. For each malicious attack, features are described as a set of punctuation and keywords (commands or functions) present in that attack. These features are labeled as 0/1 showing whether a particular keyword or punctuation is present or not in the HTTP request. We extract these features manually using the technique as explained in [25].

Table 5. Data processing

Attack types	Number of URL decoding	Characters removed and replaced
XSS	2	\n and whitespace removed
SQLi	2	\n removed
Path Traversal	4	\n removed
LDAP	1	\n removed
XPath	1	\n removed
SSI	1	\n removed and whitespaces replaced with +
OS Command	2	\n and + replaced with single whitespace
CRLFi	1	-

Table 6 shows the features used for each attack. For anomaly detection, we have considered the following features. They are `Request length, Argument's length, Number of arguments, Path length, Frequencies of Uppercase characters, Number of keywords, Frequencies of Lower case characters, Frequencies of digits[0--9], Frequencies of special characters.`

Here, the number of keywords means the count of all the keywords that occurred in the HTTP request. Keywords constitute the features of all the eight attacks. The features are extracted directly from the HTTP requests. But for the calculation of the last feature, some data processing has been done which includes 'URL decoding', lowercasing of all the characters, removal of newline characters and replacement of '+' with a single whitespace. For feature selection, information gain algorithm is applied to select the most prominent features.

Classification. For the classification of the web application attacks, we use six different machine learning classifiers namely Logistic Regression [4], K-Nearest Neighbour (KNN) [8], Support Vector Machine (SVM) [33], Decision Tree [31], Random Forest [14], Gradient Boosting Algorithm [3]. We use Python's Sckit-learn library to implement these machine learning algorithms. We carry out our experiments on Ubuntu 18.04 LTS having 32 GB RAM and Intel i7 octa-core processor. We add approximately the same number of valid requests as the number of attack requests to balance the dataset. Each attack dataset has been randomly split into 75% training and 25% testing set and the results are shown in Tables 8 and 9.

3.2 Online Implementation

In this phase, we describe the implementation of our models obtained from offline log based analysis for a live website. These models read HTTP requests in real time from the log file of a website `''security.cse.iitk.ac.in''` to detect

Table 6. Features used for different attacks

Attack types	Features used
XSS	‘&’, ‘%’, ‘/’, ‘\\’, ‘+’, “’”, ‘?’, ‘!’, ‘;’, ‘#’, ‘=’, ‘[‘, ’]’, ‘$’, ‘(‘, ’)’, ‘∧’, ‘*’, ‘,’, ‘-’, ‘<’, ‘>’, ‘@’, ‘_’, ‘:’, ‘{‘, ’}’, ‘ ’, ‘.’, ’ ’, ‘|’, ‘”’, ‘<>’, “’, ‘<>’, ‘[]’, “==”, ‘&#’, ‘document’, ‘window’, ‘iframe’, ‘location’, ‘this’, ‘onload’, ‘onerror’, ‘createelement’, ‘string.fromcharcode’, ‘search’, ‘div’, ‘img’, ‘<script’, ‘src’, ‘href’, ‘cookie’, ‘var’, ‘eval()’, ‘http’, ‘.js’
SQLi	‘–’, ‘/**/’, ‘%’, ‘+’, “’”, ‘;’, ‘#’, ‘=’, ‘[‘, ’]’, ‘(‘, ’)’, ‘∧’, ‘*’, ‘char’, ‘,’, ‘-’, ‘<’, ‘>’, ‘ ’, ‘.’, ‘|’, ‘”’, ‘<>’, ‘<=’, ‘>=’, ‘&&’, ‘||’, ‘:’, ‘!=’, ‘’, ‘count’, ‘into’, ‘or’, ‘and’, ‘not’, ‘null’, ‘select’, ‘union’, ‘#’, ‘insert’, ‘update’, ‘delete’, ‘drop’, ‘replace’, ‘all’, ‘any’, ‘from’, ‘count’, ‘user’, ‘where’, ‘sp’, ‘xp’, ‘like’, ‘exec’, ‘admin’, ‘table’, ‘sleep’, ‘commit’, ‘()’, ‘between’
Path Traversal	‘../’, ‘..\\’, ‘etc’, ‘passwd’, ‘\\.’, ‘\\/’, ‘./’, ‘/’, ‘:’, ‘//’, ‘:/’, ‘system’, ‘ini’, ‘..’, ‘exec’, ‘:\\’, ‘%00’, ‘.bat’, ‘file’, ‘windows’, ‘boot’, ‘winnt’, ‘.conf’, ‘access’, ‘log’, ‘,,’
LDAP	‘\\’, ‘*’, ,(‘, ’)’, ‘/’, ‘+’, ‘<’, ‘>’, ‘;’, ‘”’, ‘&’, ‘|’, ‘(&’, ‘(|’, ’)(‘, ’,’, ‘!’, ‘=’, ’)&’, ‘ ’, ’*)’, ’))‘, ‘&(‘, ‘+)’, ‘=)’,‘cn=’, ‘sn=’, ‘=*’, ‘(|’,‘mail’, ‘objectclass’, ‘name’
XPath	‘/∗’, ‘%’, ‘+’, ‘′’, ‘;’, ‘#’, ‘=’, ‘[’, ‘]’, ‘(’, ‘)’, ‘∧’, ‘∗’, ‘()’, ‘//’, ‘,’, ‘−’, ‘<’, ‘>’, ‘.’, ‘|’, ‘”’, ‘<>’, ‘<=’, ‘>=’, ‘&&’, ‘||’, ‘::’, ‘((’, ‘< −−’, ‘ ’, ‘or’, ‘count’, ‘path/’, ‘and’, ‘not’, ‘text()’, ‘child’, ‘position()’, ‘node()’, ‘name’, ‘user’, ‘comment’
SSI	‘<!−’, ‘−− >’, ‘#’, ‘+’, ‘,’, ‘”’, ‘etc/’, ‘/passwd’, ‘dir’, ‘#exec’, ‘cmd’, ‘fromhost’, ‘email’, ‘odbc’, ‘#include’, ‘virtual’, ‘bin/’, ‘toaddress’, ‘message’, ‘replyto’, ‘sender’, ‘#echo’, ‘httpd’, ‘access.log’, ‘var’, ‘+connect’, ‘date_gmt’, ‘+statement’, ‘log/’, ‘/mail’, ‘”mail’, ‘”id’, ‘+id’, ‘.bat’, ‘ls+’, ‘home/’, ‘winnt\\’, ‘system.ini’, ‘.conf’, ‘+-l’, ‘windows’, ‘.conf’, ‘.com’, ‘:\\’
OS Command	‘../’, ‘..\\’, ‘etc’, ‘passwd’, ‘\\.’, ‘\\/’, ‘./’, ‘:’, ‘:/’, ‘.’,‘system32’, ‘display’, ‘.exe’, ‘cmd’, ‘dir’, ‘;’, ‘tmp/’, ‘etc/passwd’, ‘wget’, ‘cat’, ‘ping’, ‘bash’, ‘ftp’, ‘|’, ‘..’, ‘exec’, ‘:\\’, ‘.bat’, ‘file’, ‘script’, ‘rm ’, ‘c:’, ‘winnt’, ‘access’, ‘log’, “’, ‘www.’, ‘http’, ‘ ’, ’bin/’, ‘telnet’, ‘echo’, ‘root’, ‘-aux’, ‘shell’, ‘uname’, ‘IP’
CRLFi	‘%0A’, ‘%0D’, ‘%0D%0A’, ‘SET’, ‘COOKIE’, ‘:’, ‘+’, ‘TAMPER’

different attacks. The same log file is fed to each attack model consecutively for detection of attacks. Figure 3 shows the architecture of our implementation.

Here, the models continuously read new requests from the log file every 15 s. Once the new requests are received, they are passed to the parser for formatting. The formatted traffic is then passed to each attack model sequentially, as shown in the Fig. 3. The attack detection models are arranged sequentially from most dangerous to least dangerous attacks [35]. Each model detects its corresponding attack and passes the rest of the traffic to the subsequent attack models for detection. Each detection process carries out its own data processing, and

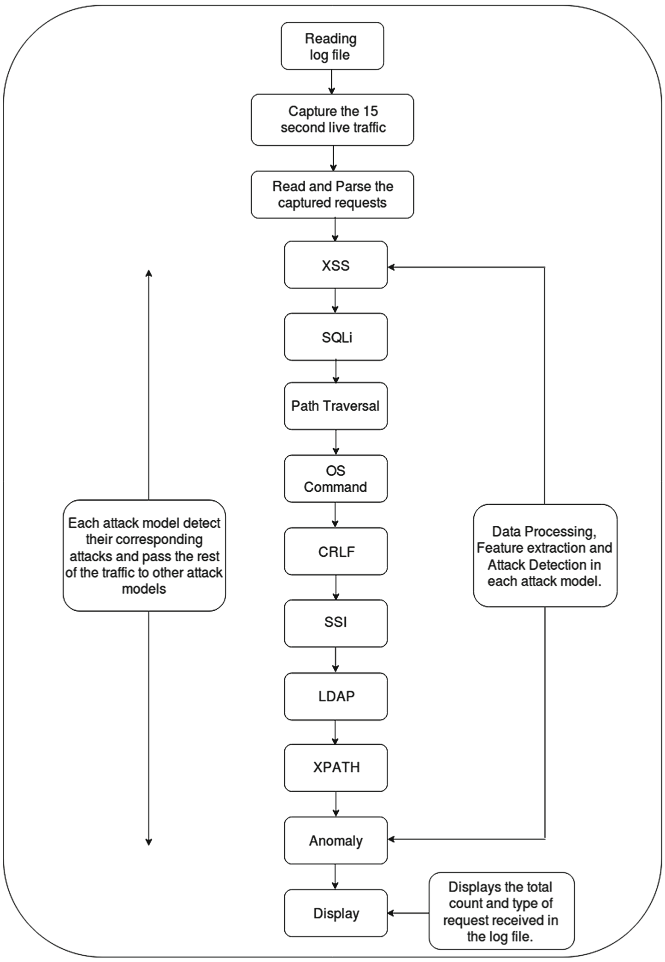

Reading log file
Capture the 15 second live traffic
Read and Parse the captured requests
XSS
SQLi
Path Traversal
OS Command
CRLF
SSI
LDAP
XPATH
Anomaly
Display
Each attack model detect their corresponding attacks and pass the rest of the traffic to other attack models
Data Processing, Feature extraction and Attack Detection in each attack model.
Displays the total count and type of request received in the log file.

Fig. 3. Architecture for Phase 2

feature extraction as both the steps are different for each attack model. After the detection of all the attacks, we display the count of attacks received and types of requests detected in the last 15 s. This procedure for detection of attack is repeated in every 15 s. Figures 4 and 5 displays screen shots of the implemented tool for our real website and the corresponding dashboard.

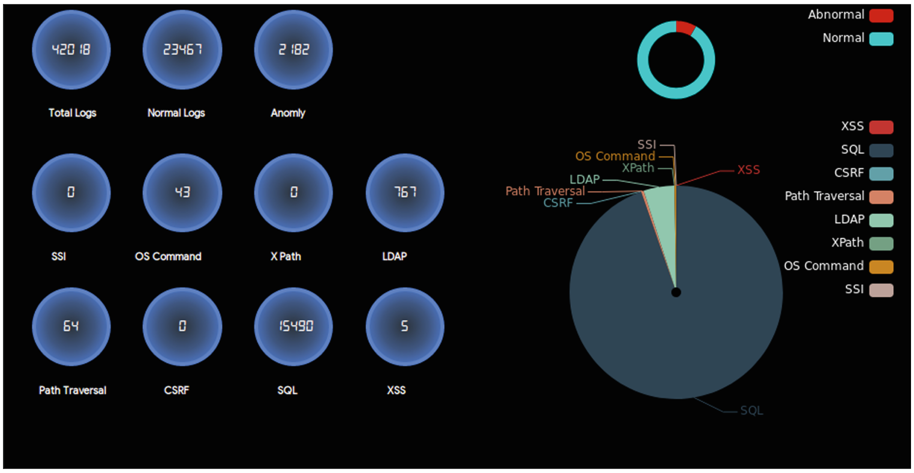
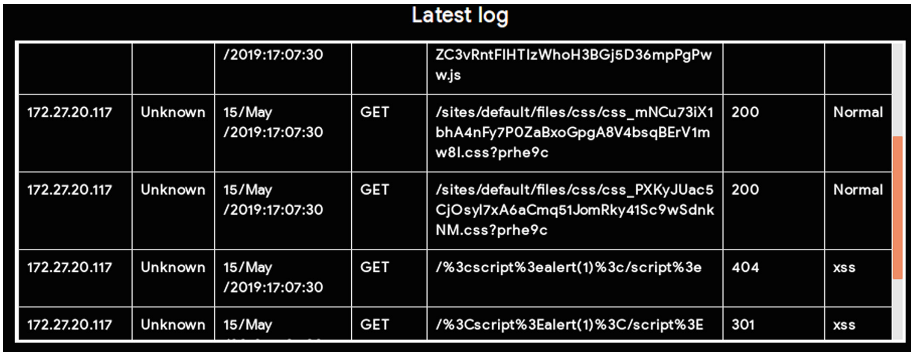


Fig. 4. IDS System showing the count of each attacks



Latest log

		/2019:17:07:30		ZC3vRntFIHTIzWhoH3BGj5D36mpPgPw w.js		
172.27.20.117	Unknown	15/May /2019:17:07:30	GET	/sites/default/files/css/css_mNCu73iX1 bhA4nFy7P0ZaBxoGpgA8V4bsqBErV1m w8l.css?prhe9c	200	Normal
172.27.20.117	Unknown	15/May /2019:17:07:30	GET	/sites/default/files/css/css_PXKyJUac5 CjOsyl7xA6aCmq51JomRky41Sc9wSdnk NM.css?prhe9c	200	Normal
172.27.20.117	Unknown	15/May /2019:17:07:30	GET	/%3cscript%3ealert(1)%3c/script%3e	404	xss
172.27.20.117	Unknown	15/May	GET	/%3Cscript%3Ealert(1)%3C/script%3E	301	xss

Fig. 5. IDS System showing the types of request received

4 Evaluation of the Proposed Framework

To evaluate the performance of our models, we use ten-fold cross-validation on each attack dataset. Table 7 shows the results of 10-fold cross-validation for different machine learning classifiers on the features selected for each attack. Tables 8 and 9 shows the performance in terms of accuracy, precision, recall, F1-score, TPR, FPR, TNR, and FNR for all the classifiers on each attack dataset. On the basis of these metrics, we select a model which performs better.

Table 8 shows the results for XSS, SQLi, Path Traversal, and LDAP attacks. For XSS attacks, Random Forest achieves the highest accuracy of 99.38% with a low FPR. Also for SQLi attacks, Random Forest performs better with an accuracy of 97.91%. Random Forest and Gradient Boosting classifiers achieve the highest accuracy of 99.28% for Path Traversal attacks. In case of LDAP attacks,

Table 7. 10-fold cross-validation results in %

Attacks\Classifiers	LR	LSVM	GB	DT	K-NN	RF
XSS	98	99	99	99	98	99
SQL	96	96	97	97	97	98
Path Traversal	97	98	98	98	95	98
OS Command	98	98	98	97	96	98
CRLF	100	100	100	100	100	100
SSI	100	100	100	100	100	100
LDAP	100	100	100	100	99	100
XPath	100	100	100	100	100	100
Anomaly	89	85	95	95	93	97

where *LR = Logistic Regression, LSVM = Linear Support Vector Machine, GB = Gradient Boosting Classifier*, *DT = Decision Tree, K-NN = K-Nearest Neighbor, and RF = Random Forest*

Table 8. XSS, SQLi, path traversal, and LDAP detection test results in %

Classifiers	XSS results							SQLi results						
	Acc.	Pre.	FS	TPR	FPR	TNR	FNR	Acc.	Pre.	FS	TPR	FPR	TNR	FNR
LR	98.04	98.4	97.98	97.55	1.4	98.6	2.45	95.7	95.16	95.84	96.53	5.1	94.9	3.47
LSVM	98.66	98.32	98.6	98.88	1.5	98.5	1.12	95.57	95	95.71	96.44	5.3	94.7	3.56
GB	98.88	98.64	98.84	99.04	1.2	98.8	0.96	96.52	95.16	96.61	98.1	5.1	94.9	1.9
DT	99.15	99.2	99.12	99.04	0.7	99.3	0.96	96.78	95.83	96.88	97.95	4.4	95.6	2.05
K-NN	99.27	99.36	99.24	99.12	0.5	99.5	0.88	96.39	95.75	96.51	97.29	4.5	95.5	2.71
RF	**99.38**	99.6	99.36	99.12	0.3	99.7	0.88	**97.91**	97	97.97	98.97	3.1	96.9	1.03
Classifiers	Path traversal results							LDAP results						
	Acc.	Pre.	FS	TPR	FPR	TNR	FNR	Acc.	Pre.	FS	TPR	FPR	TNR	FNR
LR	98.27	98.15	98.25	98.35	1.8	98.2	1.65	99.67	99.77	99.66	99.55	0.2	99.8	0.45
LSVM	98.88	99.79	98.88	97.98	0.2	99.8	2.02	99.89	100	99.88	99.77	0	100	0.23
GB	99.28	100	99.28	98.58	0	100	1.42	99.89	100	99.88	99.77	0	100	0.23
DT	99.18	100	99.18	98.38	0	100	1.62	99.89	100	99.88	99.77	0	100	0.23
K-NN	96.23	93.63	96.1	98.7	5.9	94.1	1.3	99.89	100	99.88	99.77	0	100	0.23
RF	**99.28**	100	99.28	98.58	0	100	1.42	**99.89**	100	99.88	99.77	0	100	0.23

where *Acc. = Accuracy, Pre. = Precision, FS = F1-Score, TPR = True Positive Rate, FPR = False Positive Rate, TNR = True Negative Rate, FNR = False Negative Rate*

except Logistic Regression all other classifiers achieve an highest accuracy of 99.89% having zero FPR.

The results for XPath, OS command, SSI, and anomaly detection are shown in Table 9. Random Forest classifier achieves the highest accuracy of 99.94% for detection of XPath attack with zero FPR. The highest detection accuracy for OS command attacks is 98.13% using Gradient Boosting classifier with low FPR. All the classifiers achieve the highest detection accuracy of 99.89% with zero FPR for the SSI attack. For the detection of anomaly Random Forest achieves the highest accuracy of 96.92%.

Table 9. XPath, OS command, SSI and anomaly detection test results in %

Classifiers	XPath results							OS command results						
	Acc.	Pre.	FS	TPR	FPR	TNR	FNR	Acc.	Pre.	FS	TPR	FPR	TNR	FNR
LR	99.89	100	99.9	99.8	0	100	0.2	97.85	99.01	97.75	96.53	0.9	99.1	3.47
LSVM	99.84	99.9	99.85	99.8	0.1	99.9	0.2	97.95	99.4	97.85	96.36	0.5	99.5	3.64
GB	99.89	99.9	99.9	99.9	0.1	99.9	0.1	**98.13**	99.8	98.05	96.37	0.1	99.9	3.63
DT	99.84	99.8	99.84	99.89	0.2	99.8	0.11	97.39	98.61	97.27	95.96	1.2	98.8	4.04
K-NN	99.89	99.9	99.9	99.9	0.1	99.9	0.1	96.55	98.61	96.42	94.32	1.2	98.8	5.68
RF	**99.94**	100	99.95	99.9	0	100	0.1	97.57	98.61	97.46	96.33	1.2	98.8	3.67
Classifiers	SSI results							Anomaly results						
	Acc.	Pre.	FS	TPR	FPR	TNR	FNR	Acc.	Pre.	FS	TPR	FPR	TNR	FNR
LR	99.89	100	99.89	99.79	0	100	0.21	89.13	94.43	90.58	87.03	7.7	92.3	12.97
LSVM	99.89	100	99.89	99.79	0	100	0.21	86.53	99.33	89.09	80.76	1.1	98.9	19.24
GB	99.89	100	99.89	99.79	0	100	0.21	94.96	97.54	95.54	93.62	3.2	96.8	6.38
DT	99.89	100	99.89	99.79	0	100	0.21	94.73	94.95	95.23	95.5	6.2	93.8	4.5
K-NN	99.89	100	99.89	99.79	0	100	0.21	93.2	95.14	93.94	92.77	6.2	93.8	7.23
RF	99.89	100	99.89	99.79	0	100	0.21	**96.92**	97.94	97.24	96.55	2.5	97.5	3.45

In CRLF, all the classifiers give 100% accuracy in the detection of CRLF because of the same kind of pattern among the data points.

We also check the performance of some of the attack detection models based on the data that were not used in the training, validation, and testing. For their performance comparison, we train all the classifiers on the full training dataset and test on the data which we collected from different sources [19,21] having 81804 (Attack requests: 40637 + Valid requests: 41167) requests for XSS attack detection and 9120 (Attack requests: 4824 + Valid requests: 4296) requests for anomaly detection. This experimental analysis is performed to check whether the models generalize or not, and the selected features which are used to train the model are independent of the dataset. Table 10 shows the result of the detection of XSS attacks and anomaly using all the considered machine learning classifiers, and the results show that the model performs best on the previously unseen dataset using Random Forest classifier.

Table 10. Test results of some of the attacks on new test data.

Classifiers	XSS results							Anomaly results						
	Acc.	Pre.	FS	TPR	FPR	TNR	FNR	Acc.	Pre.	FS	TPR	FPR	TNR	FNR
LR	94.03	91.34	93.9	96.62	8.3	91.7	3.38	95.42	95.11	95.14	95.17	4.3	95.7	4.83
LSVM	83.27	69.68	80.74	95.96	24	76	4.04	95.06	95.5	94.8	94.1	4	96	5.9
GB	96.79	98.01	96.85	95.73	2	98	4.27	88.27	88.26	87.64	87.03	10.5	89.5	12.97
DT	95.41	95.65	95.45	95.25	4.4	95.6	4.75	86.14	79.21	84.33	90.16	16.7	83.3	9.84
K-NN	97.19	98.86	97.25	95.7	1.1	98.9	4.3	96.99	99.33	96.88	94.57	0.6	99.4	5.43
RF	**97.53**	98.87	97.58	96.33	1.1	98.9	3.67	**97.23**	99.46	97.13	94.91	0.4	99.6	5.09

5 Analyzing the Existing Approaches

In this section, we compare our work with existing work on the detection of web attacks. In previous work, most of the attacks have been detected using regular expressions. Some of the attacks, like SQL injection, XSS, etc. are identified using Machine Learning, but they are tested on a small amount of data. To compare our approach against these, we implement the models in the earlier papers and test on our dataset. The results show that our approach performs better as compared to their approach. Table 11 summarises the results reported in their paper and the ones obtained by using their approach on our dataset.

In [26] and [27], the authors do not report their dataset details and accuracy. They provide the regular expressions used for the detection of attacks. We consider their regular expressions and apply them on our dataset and report the accuracy in Table 11. In [37], they detect many different web attacks, but the common to our approach are XSS, SQL, and Path Traversal, for which we analyse their approach on our dataset. We want to mention here that we did not implement Path Traversal attack as the corresponding regular expressions are not provided. Note that the regular expressions used to detect these attacks have become dated, and multiple rounds of encoding schemes are being used by attackers often to disguise the attack signatures. As a result, the accuracy numbers for regex based detection seems to be low compared to models trained with latest attack requests.

Table 11. Comparison with existing approaches

Authors	Approach	Reported dataset	Author's reported accuracy (%) on their dataset	Accuracy (%) on our dataset
Meyer et al. [26]	Regular expression	Not provided	Not provided	XSS: 78.44 SQL: 93.06 Path Traversal: 73.19
OWASP [27]	Regular expression	Not provided	Not provided	XSS: 49.17 SSI: 55.78 OS Command: 74.5 LDAP: 56.29
Yu et al. [37]	Regular expression	297 - XSS 113 - SQL 127 - Path Traversal	XSS - 97.98 SQL - 98.23 Path Traversal - 90.55	XSS: 78.44 SQL: 91.64 Path Traversal: Regex not provided
Althubiti et al. [13]	Machine learning	25000 - Anomaly 360000 - Normal	RF - 99.94	RF: 88.84
Mereani et al. [25]	Machine learning	13000 - XSS 13000 - Normal	K-NN - 99.75	RF: 89.81
Proposed method	Learning machine	Dataset Sect. 3.1	XSS - 99.34, SQL - 97.91 Path Traversal - 99.28, SSI - 99.89 LDAP - 99.89, XPath - 99.79 OS Command - 98.13, CRLF - 100 Anomaly - 96.92	XSS - 99.34, SQL - 97.91 Path Traversal - 99.28, SSI - 99.89 LDAP - 99.89, XPath - 99.79 OS Command - 98.13, CRLF - 100 Anomaly - 96.92

In [13] and [25] authors use machine learning approaches to detect the anomaly and XSS attack, respectively. We implement their approach to our dataset and achieve less accuracy as compared to their reported accuracy. The reason may be that the authors in [13] used a highly imbalanced dataset, but we used balanced dataset. Another reason may be that the feature set used is dependent on the dataset. This is the reason why we also implement our approach for the unseen dataset, and the results are discussed in Table 10 to test for generalization.

6 Conclusion

Web application attacks are significant threats to most of the organizations, and attacks like XSS, injections, etc. are on the rise. Hence, to detect these attacks, we implement a two-phase approach i.e. offline log-based analysis using various machine learning models and an online implementation of the log-analysis version. Here, the online log-analysis system reads HTTP requests from the log files every 15 s for the classification of these requests into corresponding attacks or normal requests. For this, the system parses those requests and then passes them to different machine learning models for detection of attacks. In this work, we found features and machine learning classifiers which are effective for high accuracy detection of attacks. Lastly, we also analyse the XSS and anomaly detection models for unseen requests. Since we use supervised machine learning algorithm; hence, we can detect only those attack requests for which we perform the analysis.

Various improvements can be done to this work. In this work, we perform the detection of web application attacks using HTTP requests, which contain only Query and Path field. This can be extended to all the other aspects such as 'Accept-Encoding,' 'Cookie,' 'Referrer', etc. We can also implement the classification of attacks into its sub-categories, e.g., XSS can be sub-classified into Stored, Reflected and DOM-based XSS. Similarly, other attacks can also be sub-categorized. In future, we plan to detect the attacks that use HTTP POST method to deliver the attack payload. As of now, we do not have any ground truth to validate live data results. Hence, in future, we shall validate these results. We also plan to develop a web application firewall using our models which would intercept HTTP requests before they are processed at the web servers.

Acknowledgement. This work has been partially supported by grants from the Science and Engineering Research Board (SERB), and Department of Science and Technology (DST), Government of India.

References

1. ECML/PKDD 2007 Dataset (2007). http://www.lirmm.fr/pkdd2007-challenge/
2. Xpath injection (2015). https://www.owasp.org/index.php/XPATH_Injection
3. Gradient boosting (2016). https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

4. Logistic regression (2016). https://machinelearningmastery.com/logistic-regression-for-machine-learning/
5. Sql injection (2016). https://www.owasp.org/index.php/SQL_Injection
6. Crlf injection (2018). https://www.owasp.org/index.php/CRLF_Injection
7. Cross-site scripting (xss) (2018). https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
8. Nearest neighbors (2018). http://scikit-learn.org/stable/modules/neighbors.html
9. Sophos India (2018).https://www.businesstoday.in/current/economy-politics/76-per-cent-indian-businesses-hit-by-cyber-attacks-in-2018-finds-survey/story/327389.html
10. World Internet Users and 2019 Population Stats (2019). https://www.internetworldstats.com/stats.htm
11. acunetix: Path traversal (2017). https://www.acunetix.com/blog/articles/path-traversal/
12. acunetix: Cross-site scripting (2019). https://www.acunetix.com/websitesecurity/cross-site-scripting/
13. Althubiti, S., Yuan, X., Esterline, A.: Analyzing http requests for web intrusion detection. KSU Proceedings on Cybersecurity Education, Research and Practice (2017)
14. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
15. Carmen Torrano, A.P., Álvarez, G.: Http csic torpeda 2012 (2012). http://www.tic.itefi.csic.es/torpeda/datasets.html
16. Carmen Torrano, A.P., Álvarez, G.: Http csic torpeda 2012 (2012). http://www.tic.itefi.csic.es/torpeda
17. Elprocus: Basic intrusion detection system (2019). https://www.elprocus.com/basic-intrusion-detection-system/
18. ENISA: Enisa threat landscape report 2018 (2019). https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018
19. Giménez, C.T., Villegas, A.P., Marañón, G.Á.: Http data set csic 2010. Information Security Institute of CSIC (Spanish Research National Council) (2010)
20. Hong Cheon, E., Huang, Z., Lee, Y.S.: Preventing sql injection attack based on machine learning. Int. J. Advancements Comput. Technol. **5**, 967–974 (2013). https://doi.org/10.4156/ijact.vol5.issue9.115
21. KF, DP: Xssed dataset (2007). http://www.xssed.com/
22. Kozik, R., Choraś, M., Renk, R., Hołubowicz, W.: Modelling http requests with regular expressions for detection of cyber attacks targeted at web applications. In: International Joint Conference SOCO 2014-CISIS 2014-ICEUTE 2014, pp. 527–535. Springer, Switzerland (2014). 10.1007/978-3-319-07995-0_52
23. Kumar, B.S., Ch, T., Raju, R.S.P., Ratnakar, M., Baba, S.D., Sudhakar, N.: Intrusion detection system-types and prevention. Int. J. Comput. Sci. Info. Tech. (IJCSIT) **4**(1), 77–82 (2013)
24. Mansfield, M.: General small business cyber security statistics (2018). https://smallbiztrends.com/2017/01/cyber-security-statistics-small-business.html
25. Mereani, F.A., Howe, J.M.: Detecting cross-site scripting attacks using machine learning. In: Hassanien, A.E., Tolba, M.F., Elhoseny, M., Mostafa, M. (eds.) AMLTA 2018. AISC, vol. 723, pp. 200–210. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74690-6_20
26. Meyer, R.: Detecting attacks on web applications from log files (2008). https://www.sans.org/reading-room/whitepapers/logging/detecting-attacks-web-applications-log-files-2074

27. OWASP: Owasp modsecurity core rule set (2014). https://github.com/SpiderLabs/owasp-modsecurity-crs/blob/master/base_rules/modsecurity_crs_40_generic_attacks.conf
28. OWASP: Owasp top 10–2017 (2017). https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
29. OWASP: Testing for ldap injection (2017). https://www.owasp.org/index.php/Testing_for_LDAP_Injection_(OTG-INPVAL-006)
30. OWASP: Command injection (2018). https://www.owasp.org/index.php/Command_Injection
31. Quinlan, R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo (1993)
32. Sarmah, A.: Intrusion detection systems: definition, need and challenges (2019). https://www.sans.org/reading-room/whitepapers/detection/intrusion-detection-systems-definition-challenges-343
33. Scholkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT press, Cambridge (2001)
34. Shatabda: Ssi injection (2018). https://medium.com/@shatabda/security-ssi-injection-what-how-fbce1dc232b9
35. Technologies, P.: Web application attack statistics (2018). https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Web-application-attacks-2018-eng.pdf
36. W3Schools: SQL Injection (2019). https://www.w3schools.com/sql/sql_injection.asp
37. Yu, J., Tao, D., Lin, Z.: A hybrid web log based intrusion detection model. In: 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 356–360. IEEE (2016)