Detecting common web attacks based on supervised machine learning using web logs

Dau Hoang
Posts and Telecommunications Institute of Technology
36 PUBLICATIONS **545** CITATIONS
SEE PROFILE

Trong Hung Nguyen
Academy of People's Security
9 PUBLICATIONS **14** CITATIONS
SEE PROFILE

Journal of Theoretical and Applied Information Technology
31st March 2021. Vol.99. No 6
© 2021 Little Lion Scientific

ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

DETECTING COMMON WEB ATTACKS BASED ON SUPERVISED MACHINE LEARNING USING WEB LOGS

[1]XUAN DAU HOANG, [2]TRONG HUNG NGUYEN

[1]Cybersecurity Lab, Posts and Telecommunications Institute of Technology, Hanoi, Vietnam

[2]Faculty of Information Technology and Security, Academy of People's Security, Hanoi, Vietnam

E-mail: [1]dauhx@ptit.edu.vn, [2]tronghungt31@gmail.com

ABSTRACT

Web attacks, such as SQLi (SQL injection) and XSS (Cross Site Scripting) have been seen critical threats to web applications, websites and web users. These types of web attacks can cause serious damages to web applications, websites and web users, ranging from bypassing authentication systems, stealing sensitive information from databases and users, to even taking the full control of server systems. To cope with web attacks, a number of methods have been researched and applied to protect web applications, websites and web users. Among them, the detection of web attacks is a promising approach in defensive layers to safeguard websites and web applications. However, some methods can only detect one kind of web attacks, while other proposals either require regular updates of detection rules, or require extensive computing resources because they use complicated detection methods. In this paper, we propose a model for web attack detection based on machine learning using web logs. Our model's main aims are (1) building the detection model automatically and without the requirement of frequent update, (2) being able to detect common types of web attacks and (3) improving the detection rate as well as lowering down the false alarm rates. The proposed detection model is built using inexpensive machine learning algorithms, including SVM, decision tree and random forest. Experiments conducted on a labelled dataset and real web logs show that the proposed model is capable of detecting common types of web attacks effectively with the highest overall detection accuracy rate of 99.68%.

Keywords: *Common Web Attacks, Web Attack Detection, SQL injection Detection, Cross Site Scripting Detection, Machine Learning-based Web Attack Detection*

1. INTRODUCTION

Web attacks, such as SQLi, XSS, CMDi (Operating System Command injection) and Path traversal have been considered constant and dangerous threats to websites, web applications and web users [1][2]. These kinds of attacks are common because of the popularity of websites and web applications and the availability of the web attack tools on the Internet [3]. We name the web attack group of SQLi, XSS, CMDi and Path traversal (Path) as "common web attacks". The major cause that allows common web attacks is the security vulnerability in the input data validation mechanisms of web systems [1][2]. Common web attacks can cause serious consequences to websites, web applications and their users. These attacks can assist attackers to bypass the web systems' authentication mechanisms, to carry out unauthorized modifications to web content and databases, to extract important data from web application databases, to steal sensitive information of web servers and web users, and even to take the full control of the web servers and/or the database servers [1][2].

Among common web attacks, SQLi or SQL injection attack is one of the most dangerous attacks to websites and web applications. SQLi is in the "Injection" web attack group that has been the first position of the Top 10 OWASP web vulnerabilities and threats for many years [1]. The main target of SQLi attacks are the databases of websites or web applications. Attackers usually exploit the vulnerabilities in websites' user input data validation to launch SQLi attacks. Malicious SQL code can be inserted into web URLs and web data input forms, then they are sent to the web server and finally executed on the database server of the web system. Figure 1 shows an example of SQLi attack to a web system, in which the attacker inserts the malicious code into the input data in order to extract the list of usernames and passwords from the database of the web system.

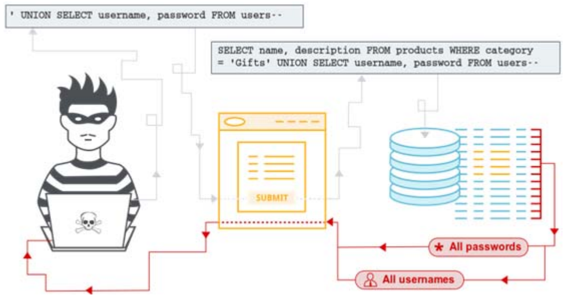

```
' UNION SELECT username, password FROM users--
```
```
SELECT name, description FROM products WHERE category
= 'Gifts' UNION SELECT username, password FROM users--
```

Figure 1: An example Of SQLi Attacks To Extract The List Of Usernames And Passwords From Web Database

XSS or Cross Site Scripting is another common type of web attacks and it is different from SQLi, where the major XSS target is the web browsers. Malicious XSS code in the form of HTML or JavaScript code is inserted into web pages and finally executed on the users' web browsers. Attackers usually use XSS to steal sensitive and valuable data stored in the user web browsers. Figure 2 illustrates a typical model of XSS attack, in which XSS code is inserted and permanently stored in the web server and then the code is executed on the user's web browser when the user visits the website.

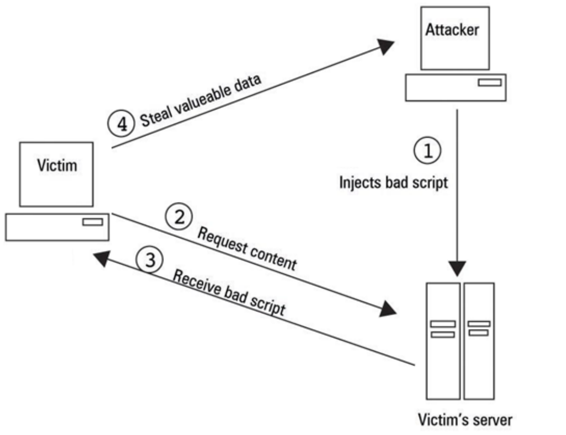
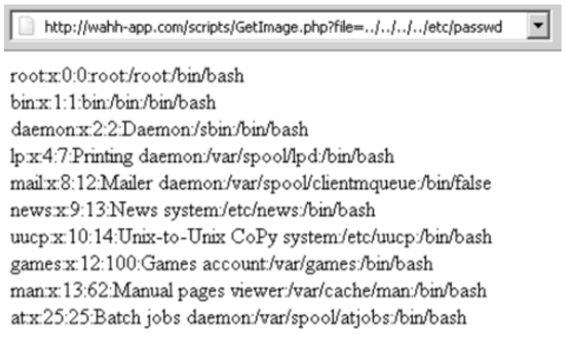


Figure 2: A Model Of XSS Attacks

CMDi or command injection is in the same "Injection" web attack group with SQLi attack. Instead of using malicious SQL code, the malicious operating system commands are inserted and executed on the server. CMDi attacks can allow attackers to execute dangerous commands, such as deleting sensitive files or folders on the web server system.

The last type of common web attacks is path traversal. This type of web attacks exploits the vulnerability in the validation of the input path strings of files or folders. The attacks allow the attackers to download the content of sensitive files of the servers. Figure 3 presents an example of path traversal attacks, in which the content of the system's password file (/etc/passwd) is retrieved and displayed on the web browser window.


```
http://wahh-app.com/scripts/GetImage.php?file=../../../../etc/passwd

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
news:x:9:13:News system:/etc/news:/bin/bash
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
games:x:12:100:Games account:/var/games:/bin/bash
man:x:13:62:Manual pages viewer:/var/cache/man:/bin/bash
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/bash
```

Figure 3: An Example Of Path Traversal Attacks

Due to the danger of common web attacks, several countermeasures have been researched and applied into practice to detect and prevent these attacks to protect websites, web applications and web users. Generally, there are 3 defensive approaches for these attacks, including (1) validate all data inputs, (2) reduce the attacking surfaces and (3) use "defense in depth" strategy [1][2]. Specifically, approach (1) requires all input data to web applications to be checked thoroughly using a set of input filters and only legitimate inputs are passed to next steps for processing. On the other hand, approach (2) requires dividing a web application into several parts and then applies suitable access controls to limit user accesses. For approach (3), several defensive measures are deployed in consecutive layers to protect websites, web applications and web users.

This paper proposes a model to detect common web attacks based on supervised machine learning methods using web logs, which belongs to approach (3). We attempt to use supervised machine learning methods to construct detection models in order to eliminate the manual construction and update of detection rules and/or signatures, as well as to increase the detection rate and to lower the false alarm rates. Based on experimental results, we select the machine learning method that gives the best overall detection performance to build the model for the web attack detection on real web logs. On the other hand, web logs generated by the web server for each hosted website by default are used as the major input to the detection model.

The remaining of our paper is structured as follows: Section 2 describes previous closely related works; Section 3 presents our proposed web attack detection model and its main processing steps; Section 4 shows experiments and results on a labelled dataset and a dataset of real web logs. Section 5 is the paper's conclusion.

2. RELATED WORKS

As mentioned in Section 1, a number of solutions of the three approaches have been researched and deployed into practice to defend again common web attacks [2]. In this section, we analyze some proposals for web attack detection, which are closely related to our work, including those in the following groups: the input data filter-based group, the anomaly-based group and the machine learning-based group.

Proposals in the input data filter-based group use sets of rules, or signatures, or techniques to filter and validate the input data in order to detect and prevent web attacks. Typical proposals in this group include OWASP Core Rule Set [5], SQLGuard [6], SQLCheck [7], SQL-IDS [8] and XSS-GUARD [9]. Core Rule Set is a set of rules developed by the OWASP project for detecting various types of web attacks in OWASP top 10 [1] with low false alarm rate. It can be used in ModSecurity [10] that is a web application firewall module attached to Apache web server. Core Rule Set is well-supported by OWASP and the web security community. However, it may be a difficulty to use Core Rule Set in some other web application firewalls or to integrate with other web servers, such as Microsoft Internet Information Services.

SQLGuard [6] and SQLCheck [7] are very similar because they both use the validation of syntax trees of SQL commands to detect SQLi attacks. Therefore, we only do a review on SQLGuard. SQLGuard is a SQLi detection and prevention system based on the validation of the syntax tree of the SQL command. SQLGuard constructs and compares the SQL command's syntax tree before inserting user input data and its syntax tree after inserting user input data. SQLGuard is able to detect SQLi attacks because the SQLi input changes the SQL command's syntax tree while the valid input does not change the SQL command's syntax tree. Experiments confirm that SQLGuard can detect SQLi attacks effectively. However, the proposed method requires the manual construction of syntax trees of all SQL valid commands of the web application. Furthermore, it requires the modification to the Java source code of the web application, which is not always possible.

Using a relatively similar approach to SQLGuard [6] and SQLCheck [7], SQL- IDS [8] is a SQLi attack detection system based on specifications. Figure 4 describes the architecture of SQL-IDS. SQL-IDS first built a set of specification rules described structures of valid SQL queries produced by the web application to be executed at the database server. Then, it monitors, pre-processes and classifies incoming SQL queries based on the pre-built rule set. Only SQL queries classified as 'valid' are forwarded to execution stage at the database server. Otherwise, invalid queries will be blocked and logged. SQL-IDS is reported to have 0% false alarm rate and it can be used to protect multiple websites because it is implemented as a proxy between the web and database servers. However, the manual building of the specification rule set is a site-specific and time-consuming task.

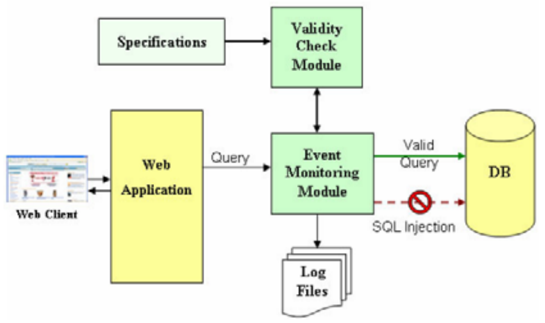


Figure 4: The Architecture Of SQL-IDS [8]

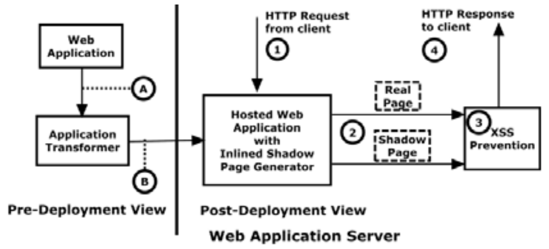


Figure 5: The Architecture Of XSS-GUARD [9]

On the other hand, XSS-GUARD [9] is a framework that monitors and prevents XSS attacks by generating a 'shadow page' and comparing it with the real page before sending the real page to client. Figure 5 describes the architecture of XSS-GUARD. The shadow web page is created in parallel with the real web page from the response of the web server, but with the clean input (without scripts) generated automatically, which has the same length as the real input. Experiments show that XSS-GUARD is able to prevent various types of XSS attacks listed by OWASP. In addition, it does not require bulky and frequent-updated rule sets. However, XSS-GUARD adds considerable loads to web servers because of the generation and comparison of shadow pages with real pages.

Web attack detection methods in the anomaly-based group first construct a 'profile' of the web application in normal working conditions and then monitor the activities of the web applications and if any significant differences between the current activities and those stored in 'profile', an attack alarm is raised. AMNESIA [10], Swaddler [12], CANDID [13] and Torrano-Gimenez et al. [14] are typical proposals in this group. AMNESIA [10] is an anomaly-based web attack detection system, in which it first scans the web application code to find and analyse all used SQL queries. Each SQL query is then modelled using the non-deterministic finite automaton (NDFA) method. The set of SQL query NDFAs is considered the 'profile' for the detection process. And then, AMNESIA monitors the web application, captures, analyses and constructs a NDFA for each SQL query sent to the database server. Next, the new SQL query's NDFA is compared to the corresponding NDFA stored in the constructed 'profile'. If a mismatch is found, the SQL query is blocked and logged. Experiments confirm that AMNESIA is able to detect all SQLi attacks in the test scenarios. However, it requires the access to the source code of the web application, which is not always possible in practice.

In the same group with AMNESIA [10], Swaddler [12] is an anomaly-based web attack detection system using a pretty different approach. Swaddler analyses a web application's internal state and learns the relationship between critical execution points of the application and its internal state to detect inconsistent, or anomalous states. Swaddler's advantages are it can be used to protect multiple websites running on the same PHP engine and the construction of the detection model can be done automatically. However, it requires to modify the PHP engine to monitor the execution flow of the web application, which can be a difficulty in the practical deployment. In addition, Swaddler's performance also needs to be taken into consideration.

On the other hand, CANDID [13] first uses dynamic analysis to extract the legitimate SQL queries of a web application in the run-time and then constructs the detection profile using the syntax tree method. Each legitimate SQL query is modelled as a syntax tree and stored into the profile. Then, it monitors the web application's SQL queries that are sent to the database system for execution. Each captured SQL query is converted to a syntax tree and then the syntax tree is compared with the profile's standard tree to look for the difference. If a mismatch is found, the SQL query is blocked and logged. CANDID's advantage over AMNESIA is it does not need to access the web application's source code. However, the proposed method only works with web applications developed and operated on Java platform.

Using a more general approach, Torrano-Gimenez et al. [14] proposes an anomaly-based detection method that can detect several types of web attacks using web traffic. The proposed method is composed of two periods, including the training period and the detection period. In the training period, normal HTTP requests to the web application are captured and features are extracted to build the profile of normal behaviour. The profile is saved into an XML file. In the detection period, each HTTP request is captured and transferred to the web access behaviour. Then, the web access behaviour is compared against the profile and any deviate from the normal behaviour is considered an attack. The proposed approach is reportedly to have a high level of detection rate as well as a low level of false alarm rate.

The machine learning-based group that includes proposals, such as Betarte et al. [15], Liang et al. [16] and Pan et al. [17], uses machine learning algorithms to construct detection models and then uses these models to detect possible web attacks. The machine learning algorithms used can be either traditional learning methods, such as naive bayes, decision tree, SVM, random forest [19][20], or deep learning methods, such as CNN and RNN [21]. Betarte et al. [15] proposes a combination of the one-class classification based on machine learning and the analysis based on n-gram technique to improve ModSecurity's detection performance. The one-class classification is used when only normal data for training is available and the n-gram analysis is used when both normal and attacked data for training are available. Experiments show that the proposed method outperforms ModSecurity's OWASP Core Rule Set [5].

Liang et al. [16] proposes to use the RNN deep learning to build the web attack detection models, as shown in Figure 6. Experiments on the CSIC 2010 dataset [22] confirm that the proposed approach has the accuracy of over 98% on overall. In addition, the proposed method can eliminate the manual and time-consuming task of the feature selection and extraction. On the other hand, Pan et al. [17] proposes the Robust Software Modelling

Tool to monitor and extract runtime information of an application and then use collected information to train the stacked denoising autoencoder to build the detection model. Experiments show that the proposed approach can detect various types of web attacks and achieves the F1-score of over 91% on average.

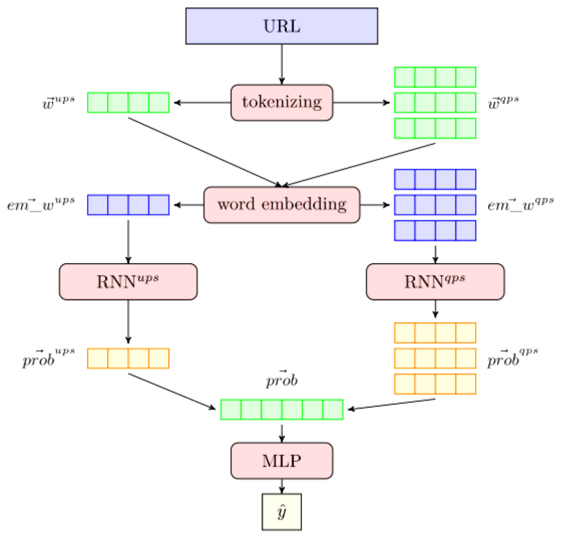


Figure 6: The Architecture Of Liang et al. [16]

From the above reviews, we can draw some comments as the following:

– Proposals in the input data filter-based group, such as OWASP Core Rule Set [5], SQLGuard [6], SQLCheck [7] and SQL-IDS [8] can detect web attacks effectively. However, they require the manual construction and frequent update to the detection rules or models;
– SQLGuard [6], SQLCheck [7], SQL-IDS [8] and XSS-GUARD [9] can only detect one type of web attacks, in which SQLGuard [6], SQLCheck [7], SQL-IDS [8] can only detect SQLi attacks while XSS-GUARD [9] can only detect XSS attacks;
– XSS-GUARD [9] faces the server's performance degradation problem because of the generation and comparison of a shadow page to the real web page for every user request. Similarly, Pan et al. [17] also faces the server's performance degradation problem because of using Robust Software Modelling Tool to monitor the server's real-time execution;
– Solutions in the machine learning-based group, such as Liang et al. [16] and Pan et al. [17] use deep learning for web attack detection. This is a relatively new approach in the field. However deep learning is generally expensive and it may not be suitable for real-time web attack detection. In addition, their detection performance is lower (Pan et al. [17]), or only slightly higher than that of the traditional supervised machine learning (Liang et al. [16]).

In summary, the research issues of existing works include (1) manual construction and frequent update to the detection rules or models, (2) only able to detect a single type of web attacks, (3) requirement of extensive computing resources and (4) not high detection rate. In order to address the above research issues, we propose a model to detect common web attacks based on supervised machine learning using web logs with the following advantages over previous works:

– The process of feature selection, extraction and model construction can be done automatically in the training stage. Therefore, our detection model does not require frequent updates;
– Our model can detect 4 major types of web attacks, including SQLi, XSS, CMDi and Path traversal;
– Inexpensive traditional machine learning algorithms, including SVM, decision tree and random forest are used to achieve a high detection performance. This allows our model to require less computing resource for the model construction and for the classification of web logs to detect web attacks;
– Our detection model performs better than those in the literature with the overall detection accuracy rate of 99.68%.
– The data input of the detection model are web logs, which are available by default on most modern web servers. This means the data collection is fairly simple and this makes it easier for the practical deployment.

3. MACHINE LEARNING-BASED MODEL FOR DETECTING COMMON WEB ATTACKS

3.1 The Proposed Model

Our model for web attack detection is composed of the training stage and the detection stage. The training stage as illustrated in Figure 7 consists of 3 steps as follows:

1. Data collection for training: Normal URIs (Uniform Resource Identifier) and attacked URIs are collected to form the dataset for training;
2. Data pre-processing: The collected dataset is pre-processed to extract classification features, in which each URI is converted into a feature

vector. The result of this step is a M×(N+1) training matrix, where M is the number of data items and N is the number of the classification features. The values in the matrix's last column store the labels of data items;

3. The training matrix is put into the 'Training' step to construct the Classifier or the Model that is used in next stage.

The detection stage as presented in Figure 8 also includes 3 steps as follows:

1. URI extraction: URIs are extracted from web logs and each URI is the input of the detection process in sequence;
2. URI pre-processing: The URI is pre-processed using the same method done for each URI of the training set. The output of this step is a URI feature vector, which is used in the next step;
3. URI classification: The constructed Classifier/Model is used to classify the URI's feature vector. This step's output is the predicted label for the URI. The 'Normal' label is for the normal URI and 'Attacked' label is for the the web attack URI.

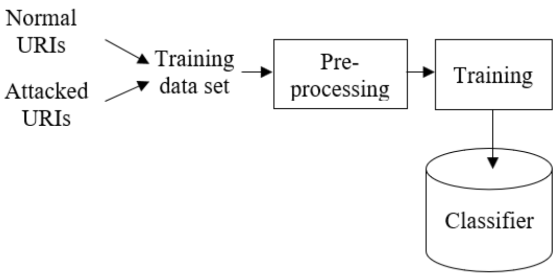


Figure 7: Proposed Model For Web Attack Detection: Training Stage

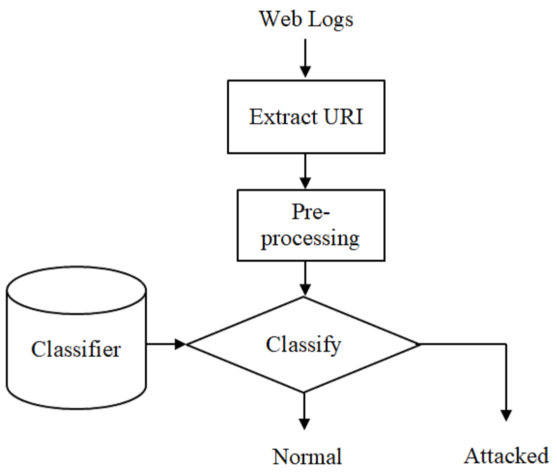


Figure 8: Proposed Model For Web Attack Detection: Detection Stage

3.2 Data Pre-processing

The two tasks of URI feature extraction and vectorization in the pre-processing step are carried out as follows:

- The extraction of URI features based on the n-gram technique. The n-gram technique is selected because of its simplicity and fast execution. Specifically, the 3-gram is chosen for URI feature extraction.
- The vectorization of URI features based on the combination of Term Frequency and Inverse Document Frequency (TF-IDF) methods [18]. The tf-idf value of each 3-gram is computed using the following formulas:

$$\mathrm{tf}(t,d) = \frac{\mathrm{f}(t,d)}{\max\{\mathrm{f}(w,d) : w \in d\}} \quad (1)$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \quad (2)$$

$$\text{tf-idf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D) \quad (3)$$

where tf(t, d) is the frequency of 3-gram t in URI d; f(t, d) is the occurrence number of 3-gram t in URI d; max{f(w,d):w∈d} is the maximum occurrence number of any 3-gram in URI d; D is the set of all URIs and N is the URI total number.

Since the number of URI classification features is pretty large, the Principle Component Analysis method is used to lower the feature number to 256, which is selected based on empirical.

3.3 Training and Cross-Validation

The training step uses some common supervised machine learning algorithms supported by Sklearn library in Python to construct and validate the detection models. The machine learning algorithms used include naive bayes, SVM, decision tree and random forest. For each algorithm, we randomly take 80% of the dataset for training to construct the detection model, and then use 20% of the dataset for validation to get the performance measurements. The final performance measurements are computed as the average of the measurements of 10 runs. In the end, the learning method that gives the best performance on overall is selected for the construction of the model to detect web attacks from real web logs.

We use 6 measurements, including TPR (True Positive Rate or Recall), FPR (False Positive Rate), FNR (False Negative Rate), PPV (Positive Predictive Value or Precision), F1 (F1-Score) and ACC (Overall Accuracy) to measure the proposed model's performance as the following:

$$PPV = \frac{TP}{TP + FP} \quad (4)$$

$$TPR \frac{TP}{TP + FN} \quad (5)$$

$$FPR = \frac{FP}{FP + TN} \quad (6)$$

$$FNR = \frac{FN}{FN + TP} \quad (7)$$

$$F1 = \frac{2TP}{2TP + FP + FN} \quad (8)$$

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} \quad (9)$$

where TP, FP, FN and TN are elements of the confusion matrix given in Table 1.

Table 1: TP, FP, FN And TN In The Confusion Matrix

		Actual Class	
		Attacked	*Normal*
Predicted Class	*Attacked*	TP (True Positives)	FP (False Positives)
	Normal	FN (False Negatives)	TN (True Negatives)

In addition, we use the Detection Rate (DR) to measure the effectiveness of the proposed detection model for each type of web attack. The DR for each web attack type is calculated the following:

$$DR = \frac{Number\ of\ detected\ attacks}{Total\ Number\ of\ attacks} \quad (10)$$

3.2 Overview of Machine Learning and Its Techniques

3.2.1 Overview of Machine Learning

Machine learning is a field of computer science, which involves the study and construction of techniques that enable computers to self-study based on the input data to solve specific problems [19][20]. Based on the learning methods, machine learning techniques are usually classified into three main categories: supervised learning, unsupervised learning and semi-supervised learning [19]. In supervised learning, labelled training data is used to train the classifier. The machine will "learn" from the labelled patterns to construct the classifier and use it to predict labels for new data. On the other hand, unsupervised learning uses unlabelled training data as the input. In this form, the machine will "learn" by analyzing the data characteristics to build the classifier. Semi-supervised learning is the combination approach between supervised and unsupervised learning. In semi-supervised learning, the input training dataset will contain both labelled and unlabelled data. Each method has its own advantages and disadvantages and has its own application domain. In this paper, we only examine the effectiveness of supervised learning techniques in web attack detection and the next subsection briefly describes some of the common supervised machine learning algorithms [19][20], including Naive Bayes, Support Vector Machine, decision tree, and random forest.

3.2.2 Supervised Learning Techniques

a. Naïve Bayes

Naive Bayes is a conditional probability model that is based on the Bayes theorem [16]. In the problem of classification, the data consists of: *D* is the set of training data that has been vectorized as $\vec{x} = (x_1, x_2,..., x_n)$, C_i is subclass *i*, with $i = 1,2, ..., m\}$. Assume that all properties are conditionally independent of each pair together. According to Bayes theorem, we have:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \quad (11)$$

Based on the conditionally independent characteristic, we have:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \quad (12)$$

where, $P(C_i|X)$ is the probability of class *i* given sample *X*, $P(C_i)$ is the probability of class *i* and $P(x_k|C_i)$ is the probability of the *k* property that has the value of x_k given *X* in class *i*.

b. Support Vector Machine

Support Vector Machine (SVM) introduced by Vapnik and his colleagues in 1992 is in the class of supervised learning algorithms. SVM has been widely used for classification and regression analysis. The SVM's main idea is to consider the input data as points in the *n*-dimensional space and from the initial labelled training data to find a hyperplane that accurately categorizes these data points. The resulting hyperplane is then used to classify the new unlabelled data [19].

In practice, it is possible to generate multiple different hyperplanes using the original data for classifying new data. However, the problem is that in an *n*-dimensional space with such sample data sets, how to find a hyperplane to always ensure that it best divides the data points. It can be understood that a good hyperplane is the hyperplane, whose distance from the classified data points closest to it is largest. The equation containing these data points is called the margins, so in other words a good hyperplane is the hyperplane with the distance between it and the margin as large as possible [19].

c. Decision Tree

Decision tree is a predictive model that has been used widely to solve the classification problem [19][20]. Decision tree creates models that allow the object classification by creating a set of decision rules. These rules are extracted based on the set of characteristics of the training data. In a decision tree, leaves represent classes and each child node in the tree and its branches represent a combination of features that lead to classification. Therefore, classifying an object will begin with checking the value of the root node, and then continuing downward under the tree branches corresponding to those values. This process is performed repeatedly for each node until it cannot go any further and touch the leaf node. For the best model, the decision to select the root node and sub-node while building the decision tree is based on the Information Gain (IG) and Gini Impurity measurements [19]. The decision tree algorithm used for experiments in this paper is the CART tree supported by the Python Sklearn library.

d. Random Forest

Random Forest (RF) is a chain member of decision tree algorithms. The random forest's idea is to create some decision trees. These decision trees will run and produce independent results. The answer predicted by the largest number of decision trees will be chosen by the random forest [19]. In order to ensure that the decision trees are not the same, random forest randomly selects a subset of the characteristics of each node. The remaining parameters are used in the random forest as those in the decision trees.

4. EXPERIMENTS AND RESULTS

4.1 Experiments on HTTP Param Dataset

4.1.1 The HTTP Param Dataset

The HTTP Param Dataset [23] consists of 31,067 URI payloads of web requests, including the payload lengths and labels. There are 2 payload labels: Norm (no attack) and Anom (attack). Anom payloads in turn are divided into 4 types: SQLi, XSS, CMDi and Path-traversal. Figure 9 provides some samples of the HTTP Param Dataset. The amount of each type of payloads is as follows:

- 19,304 normal payloads;
- 10,852 SQLi payloads;
- 532 XSS payloads;
- 89 CMDi payloads;
- 290 path traversal payloads.

The length of payloads is varied, which ranges from 1 to 1058 characters. On the other side, the HTTP Param Dataset is stored in some files according to the usage purposes:

- A single file that consists of the full dataset;
- Two files, in which the training file includes 20,000 payloads and the testing file includes of 11,067 payloads.

In our experiments, we use the single file of the full dataset for training and validating the proposed model, in which the 10-fold cross-validation method is used with 80% of dataset for training and 20% of dataset for cross-validation.

4.1.2 Detection Model's Performance Results

As mentioned in Section 3.3, we use the cross-validation technique to get the performance of the proposed web attack detection model. The most common supervised machine learning methods, such as naive bayes, SVM (Linear and RBF kernels), decision tree and random forest have been used to construct and validate the proposed model. The ratio between the amounts of data for training and validating is 80:20. The final performance measurements are the average of 10 runs' results for each machine learning algorithm. Table 2 gives the proposed model's 10-fold cross-validation confusion matrix parameters (TP, FP, FN and TN) on average. Table 3 shows the overall performance values (PPV, FPR, TPR, FNR, ACC and F1) for all types of web attacks. On the other hand, Table 4 provides the detection rates (DR) for each type of web attacks as well as the average of all web attack types.

4.2 Experiments on real web logs

In this section, we provide experimental results on real web logs collected from our web servers. Generally, common web servers, such as Mozilla Apache, Microsoft IIS and Ng ngix can create logs for each website they host on web requests from users. Most web logs are in the form of plain text and a text line is a log record. Although there have been many log formats, the W3C Extended log file format [24] is most widely used and supported by most modern web servers. Figure 10 shows a part of web log file created by Microsoft IIS using the W3C Extended log file format.

Based on the performance evaluation done in Section 4.1, we selected the model created using the random forest machine learning algorithm to be the 'Classifier' for classifying the real web logs to detect web attacks in this section. From the web log file, we extract each access URI that is a combination of cs-uri-stem and cs-uri-query fields of a log record. Then the URI is put into the detection stage as described on Figure 4. Experiments on real web logs confirm that our proposed model is able to detect common web attacks correctly and efficiently. Table 5 shows typical attacked and normal payloads detected using real web logs.

4.3 Discussion

In this sub-section, we discuss the detection performance of the proposed model on three aspects: (1) the effect of the distribution of web attack payloads to the detection rate, (2) the detection performance on various machine learning algorithms and (3) the comparison between the proposed model and previous related proposals.

As mentioned in Section 4.1.1, the HTTP Param Dataset [23] is not balanced because the normal and SQLi payloads dominate the dataset with about 97% of the all payloads. This is reasonable because SQLi is the most common type of web attack. Other types of web attacks, including XSS, CMDi and Path Traversal are less common than SQLi and they only account for 3% of all payloads. The unbalanced amount of web attack payloads affects heavily on the detection performance for each type of web attacks. The detection rate for SQLi is highest and detection rate for CMDi is lowest. Specifically, the detection rates for SQLi, Path traversal, XSS and CMDi using random forest algorithm are 99.51%, 97,78%, 88.74% and 73.89%, respectively, according to the performance results give in Table 4. The detection rate of CMDi attacks is not high because the amount of training data for this type of web attacks is not sufficient to build a good detection model.

Regarding the proposed model's detection performance on machine learning algorithms, the random forest performs best and naive bayes performs worst, according to the results given in Table 3. The model's performance using decision tree and SVM is almost at the same level. The model's F1-scores based on random forest, decision tree, Rbf-SVM, Linear-SVM and naive bayes are 99.57%, 98.76%, 98.33%, 98.10% and 75.10%, respectively. Although random forest requires more computational resources than that of decision tree, it performs much better than decision tree, especially on reducing the false alarm rates. The FPR and FNR produced by random forest-based model are 0.0775% and 0.7253% while the results by decision tree-based model are 0.5185% and 1.6122%, respectively. Therefore, we select random forest-based model as the final model for detecting attacks on real web logs.

As mentioned in Section 3.1, our detection model has been built using supervised machine learning algorithms from the training data automatically and therefore it does not require the manual construction and frequent update of the rule set as required in [5][6][7][8][9]. In addition, our model does not require the access to the source code of web applications (as required in [11]), nor it needs special mechanisms (as required in [17]) to get the input information because it uses web logs as the input to detect the web attacks. Regarding the detection performance, our model performs better than the highest accuracy of 98.42% of Liang et al. [16] and much better than that of 91.40% of Pan et al. [17], which use expensive deep learning techniques for web attack detection, as shown in Table 6.

Table 6. The Overall Detection Accuracy Of Our Model And Liang et al. [16] And Pan et al. [17]

Our model	Liang et al. [16]	Pan et al. [17]
99.68%	98.42%	91.40%

5. CONCLUSION

In this paper, we propose a model for detecting web attacks, which is based on supervised machine learning methods using web logs. The proposed model is capable of detecting four types of the most dangerous web attacks, including SQLi, XSS, CMDi and path traversal. Experiments on the labelled data set and real web logs confirm that our random forest-based model achieves high overall accuracy (ACC) and F1-score of 99.68% and 99.57%, respectively. The performance results confirm that our detection model is able to detect common web attacks effectively and it performs better than state-of-the-art detection models based on deep learning [16][17].

Apart from the high detection performance, the proposed model has some additional advantages over previous proposals: (1) it is built using inexpensive supervised machine learning algorithms to gain a good detection performance. This is important for practical deployment because a light-weight web attack detection system usually requires less computing resource to process a large amount of web logs; and (2) the model's construction can be done automatically and it does not require the frequent update.

For future work, we will extend the proposed model so that it can detect more types of web attacks. The new model should have higher detection rates for some special types of web attacks, including XSS and CMDi.

REFRENCES:

[1] OWASP Project, https://owasp.org, last accessed 2020/09/20.

[2] A.K. Baranwal, "Approaches to detect SQL injection and XSS in web applications," *EECE 571B, Term Survey Paper*, University of British Columbia, Canada, April 2012.

[3] Website Attack Tools, https://sourcedefense.com/glossary/website-attack-tools/, last accessed 2020/06/20.

[4] A. Tajpour, S. Ibrahim and M. Masrom, "SQL Injection Detection and Prevention Techniques," *International Journal of Advancements in Computing Technology*, vol. 3, no. 7, August 2011. DOI:10.4156/IJACT.VOL3.ISSUE7.11.

[5] OWASP ModSecurity Core Rule Set, https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project, last accessed 2020/09/20.

[6] G.T. Buehrer, B.W. Weide and P.A.G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," *Proceedings of the 5th international workshop on Software engineering and middleware*, September 2005, pp. 106–113.

[7] Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Applications," *ACM SIGPLAN Notices*, vol. 41, pp. 372-382, 2006.

[8] K. Kemalis, T. Tzouramanis, "SQL-IDS: A Specification-based Approach for SQL injection Detection," *ACM SAC'08*, pp. 2153-2158, March 16-20, 2008, Fortaleza, Brazil.

[9] P. Bisht and V.N. Venkatakrishnan, "XSS-GUARD: Precise dynamic prevention of Cross-Site Scripting Attacks," *In Proceeding of 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, LNCS 5137, pp. 23-43, 2008.

[10] Mod Security, https://www.modsecurity.org, last accessed 2020/09/20.

[11] W.G.J. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks," *IEEE and ACM International Conference on Automated Software Engineering*, pp.174-183, November 2005, USA.

[12] M. Cova, D. Balzarotti, "Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications," *International Symposium on Recent Advances in Intrusion Detection*, Vol. 4637, pp. 63-86, 2007.

[13] P. Bisht, P. Madhusudan and and V.N. Venkatakrishnan, "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks," *ACM Transactions on Information and System Security*, vol. 13, March 2010.

[14] C. Torrano-Gimenez, A. Pérez-Villegas and G. Alvarez, "An Anomaly-Based Approach for Intrusion Detection in Web Traffic," *The Allen Institute for Artificial Intelligence*, 2009.

[15] G. Betarte, E. Giménez, R. Martínez, and A. Pardo, "Machine learning-assisted virtual patching of web applications," [Online] https://arxiv.org/abs/1803.05529, Mar 2018.

[16] J. Liang, W. Zhao, W. Ye, "Anomaly-Based Web Attack Detection: A Deep Learning Approach," *ICNCC 2017*, pp. 80-85, December 8–10, 2017, Kunming, China.

[17] Y. Pan, F. Sun, Z. Teng, J. White, D.C. Schmidt, J. Staples and L. Krause, "Detecting web attacks with end-to-end deep learning," *Journal of Internet Services and Applications*, vol. 10:16, SpringerOpen, 2019.

[18] H. Wu, R. Luk, K. Wong and K. Kwok, "Interpreting TF-IDF term weights as making relevance decisions," *ACM Transactions on Information Systems*, vol. 26, no.3, 2008.

[19] A. Smola and S.V.N. Vishwanathan, "Introduction to Machine Learning," Cambridge University, 2008.

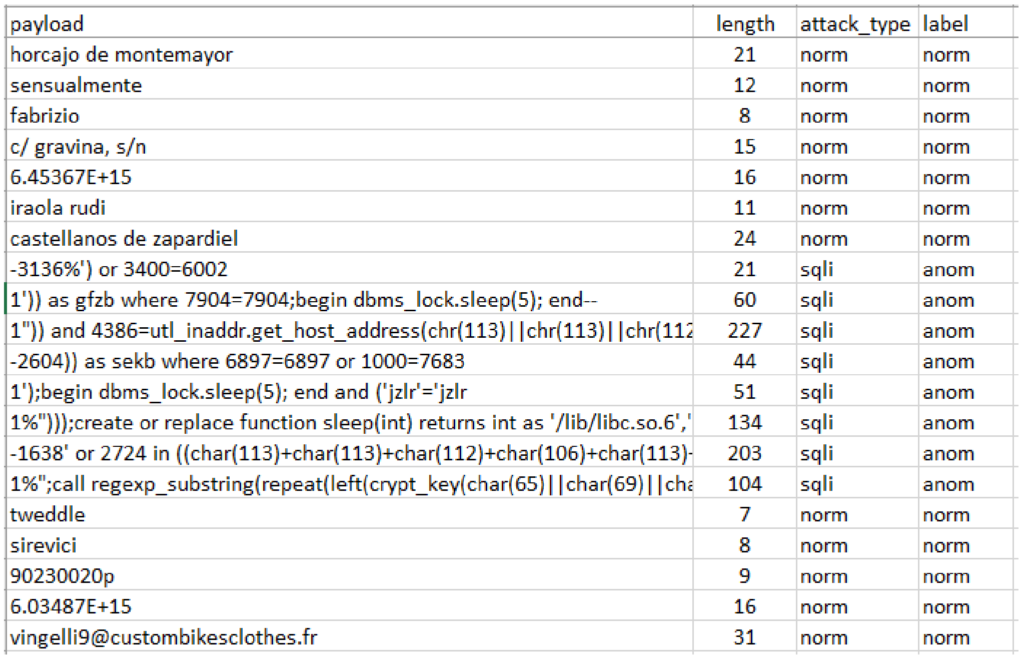
[20] N.K. Sangani, H. Zarger, "Machine Learning in Application Security," *Book chapter in "Advances in Security in Computing and Communications"*, IntechOpen, 2017.

[21] Towards Data Science, https://towardsdatascience.com/an-introduction-to-deep-learning- af63448c122c, last accessed 2020/09/20.

[22] HTTP DATASET CSIC 2010, https://www.isi.csic.es/dataset/, last accessed 2020/09/20.

[23] HTTP Param Dataset, https://github.com/Morzeux/HttpParamsDataset, last accessed 2020/09/20.

[24] Extended Log File Format, https://www.w3.org/TR/WD-logfile.html, last accessed 2020/09/20.



payload	length	attack_type	label
horcajo de montemayor	21	norm	norm
sensualmente	12	norm	norm
fabrizio	8	norm	norm
c/ gravina, s/n	15	norm	norm
6.45367E+15	16	norm	norm
iraola rudi	11	norm	norm
castellanos de zapardiel	24	norm	norm
-3136%') or 3400=6002	21	sqli	anom
1')) as gfzb where 7904=7904;begin dbms_lock.sleep(5); end--	60	sqli	anom
1")) and 4386=utl_inaddr.get_host_address(chr(113)||chr(113)||chr(112	227	sqli	anom
-2604)) as sekb where 6897=6897 or 1000=7683	44	sqli	anom
1');begin dbms_lock.sleep(5); end and ('jzlr'='jzlr	51	sqli	anom
1%")));create or replace function sleep(int) returns int as '/lib/libc.so.6','	134	sqli	anom
-1638' or 2724 in ((char(113)+char(113)+char(112)+char(106)+char(113)-	203	sqli	anom
1%";call regexp_substring(repeat(left(crypt_key(char(65)||char(69)||cha	104	sqli	anom
tweddle	7	norm	norm
sirevici	8	norm	norm
90230020p	9	norm	norm
6.03487E+15	16	norm	norm
vingelli9@custombikesclothes.fr	31	norm	norm

Figure 6: Sample Records Of The HTTP Param Dataset [23]

Table 2. Average of TP, FP, FN And TN On 10-Fold Cross-Validation

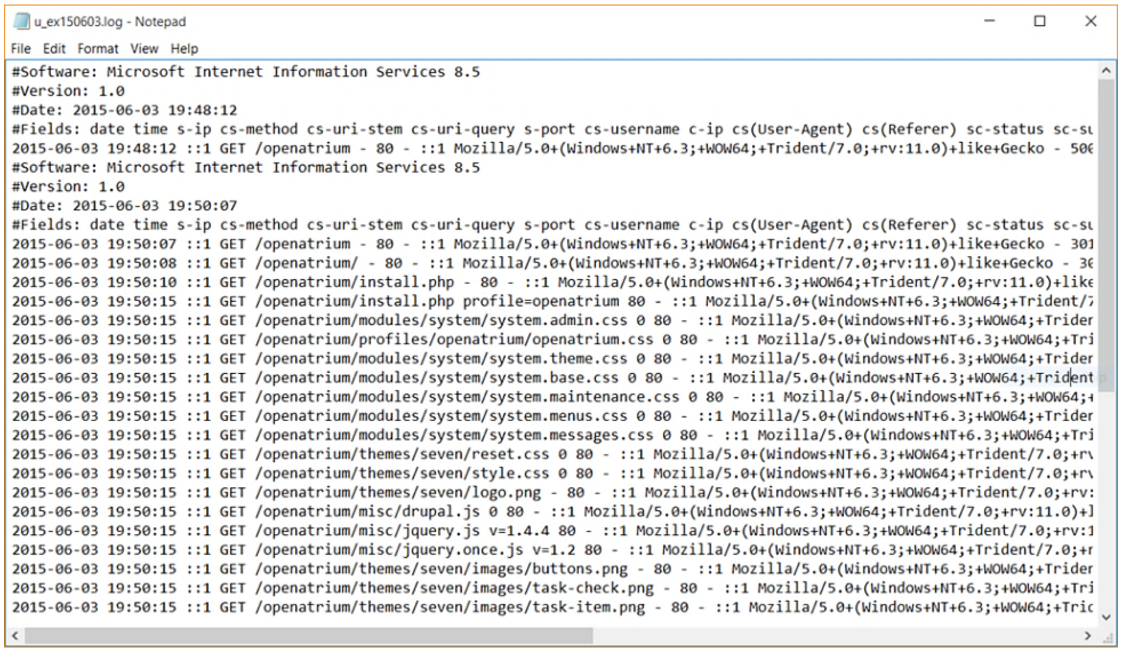
Algorithms	TP	FP	FN	TN
Naive bayes	2184	1315	133	2582
Linear-SVM	2322	31	48	3813
Rbf-SVM	2319	20	38	3837
Decision Tree	2320	56	34	3804
Random Forest	2327	3	17	3867

Table 3. The Overall Performance On Machine Learning Algorithms

Algorithms	PPV (%)	TPR (%)	FPR (%)	FNR (%)	ACC (%)	F1 (%)
Naive bayes	62.42	94.26	33.74	5.7402	76.70	75.10
Linear-SVM	97.64	98.56	1.4508	1.4444	98.55	98.10
Rbf-SVM	98.68	97.97	0.8065	2.0253	98.73	98.33
Decision Tree	99.14	98.39	0.5185	1.6122	99.07	98.76
Random Forest	99.87	99.27	0.0775	0.7253	99.68	99.57

Table 4. Detection Rates (DR) For Web Attacks On Machine Learning Algorithms

Algorithms	SQLi (%)	XSS (%)	CMDi (%)	Path (%)	Average (%)
Naive bayes	94.46	91.42	50.85	96.91	94.05
Linear-SVM	99.23	87.05	77.35	97.18	98.47
Rbf-SVM	99.32	69.27	32.34	91.25	97.28
Decision Tree	99.49	86.31	23.66	92.60	98.11
Random Forest	99.51	88.74	73.89	97.78	98.78



```
#Software: Microsoft Internet Information Services 8.5
#Version: 1.0
#Date: 2015-06-03 19:48:12
#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent) cs(Referer) sc-status sc-s
2015-06-03 19:48:12 ::1 GET /openatrium - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+rv:11.0)+like+Gecko - 50
#Software: Microsoft Internet Information Services 8.5
#Version: 1.0
#Date: 2015-06-03 19:50:07
#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent) cs(Referer) sc-status sc-s
2015-06-03 19:50:07 ::1 GET /openatrium - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+rv:11.0)+like+Gecko - 301
2015-06-03 19:50:08 ::1 GET /openatrium/ - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+rv:11.0)+like+Gecko - 3
2015-06-03 19:50:10 ::1 GET /openatrium/install.php - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+rv:11.0)+like
2015-06-03 19:50:15 ::1 GET /openatrium/install.php profile=openatrium 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/
2015-06-03 19:50:15 ::1 GET /openatrium/modules/system/system.admin.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trider
2015-06-03 19:50:15 ::1 GET /openatrium/profiles/openatrium/openatrium.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Tri
2015-06-03 19:50:15 ::1 GET /openatrium/modules/system/system.theme.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trider
2015-06-03 19:50:15 ::1 GET /openatrium/modules/system/system.base.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident
2015-06-03 19:50:15 ::1 GET /openatrium/modules/system/system.maintenance.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;
2015-06-03 19:50:15 ::1 GET /openatrium/modules/system/system.menus.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trider
2015-06-03 19:50:15 ::1 GET /openatrium/modules/system/system.messages.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Tri
2015-06-03 19:50:15 ::1 GET /openatrium/themes/seven/reset.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+r
2015-06-03 19:50:15 ::1 GET /openatrium/themes/seven/style.css 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+r
2015-06-03 19:50:15 ::1 GET /openatrium/themes/seven/logo.png - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+rv:
2015-06-03 19:50:15 ::1 GET /openatrium/misc/drupal.js 0 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+rv:11.0)+l
2015-06-03 19:50:15 ::1 GET /openatrium/misc/jquery.js v=1.4.4 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+rv:1
2015-06-03 19:50:15 ::1 GET /openatrium/misc/jquery.once.js v=1.2 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trident/7.0;+r
2015-06-03 19:50:15 ::1 GET /openatrium/themes/seven/images/buttons.png - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Trider
2015-06-03 19:50:15 ::1 GET /openatrium/themes/seven/images/task-check.png - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Tri
2015-06-03 19:50:15 ::1 GET /openatrium/themes/seven/images/task-item.png - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+Tric
```

Figure 7: A Part Of Web Log File Using W3C Extended Log File Format

Table 5: Typical Attacked/Normal Payloads Detected Using Real Web Logs

Detection label	Attack payload used
Path Traversal	/./././././././././etc/passwd
SQLi	1' where 2145=2145;select sleep(5)#d
XSS	<bgsound src='javascript:document.cookie=true;'>
Path Traversal	EXEC xp_cmdshell('cat ../../../etc/passwd')#
SQLi	tabid=109' and 'x'='y
SQLi	"-7868"" union all select 1805--"
SQLi	tabid=439) AND 9999=CAST((CHR(113)||CHR(106)||CHR(113)|| CHR(112)||CHR(113))
SQLi	"1%""))) and (select * from (select(sleep(5)))gcrr)#"
Normal	tabid=471&language=en-US
Path Traversal	sname=../../../../../../../../../../../../../../../../Windows/system.ini
Path Traversal	name=../../../etc/passwd&sid=1293&pageid=4321
SQLi	topicid=82&pageid=61669 union select /**/ unhex(hex(version()))
SQLi	topicid=82&pageid=6166 or (1, 2) = (select * from (select name_const(CHAR(111, 108, 111, 108, 111, 115, 104, 101, 114), 1)
CMDi	"& ping -n 30 127.0.0.1 &"
Path Traversal	txtSearchNews=.../.\.../.\.../.\.../.\.../.\.../.\.../.\.../.\.../.\windows/win.ini
Path Traversal	id=../../../../../../../../../../../../../../../../Windows/system.ini&txtSearchNews=ZAP
CMDi	"<!--#exec cmd=""/bin/cat /etc/shadow""-->","39"
Path Traversal	query=/../WEB-INF/web.xml
XSS	"<img id=xss src="" javascript:alert('xss');"">"
XSS	sname="><script > alert(String.fromCharCode(88, 83, 83)) </script> &sid=1300&pageid=32782