Applying and Evaluating Supervised Learning Classification Techniques to Detect Attacks on Web Applications

Madduri Venkata Sai Soma Manish, Rajesh Kannan Megalingam

Abstract*: Web applications are the source of information such as usernames, passwords, personally identifiable information, etc., they act as platforms of knowledge, resource sharing, digital transactions, digital ledgers, etc., and have been a target for attackers. In recent years reports say that there is a spike in the attacks on web applications, especially attacks like SQL injection and Cross Site Scripting have grown in drastic numbers due to discovery of new vulnerabilities. The attacks on web applications still persist due to the nature of attack payloads, as these payloads are highly heterogeneous and look very similar to regular text even web applications with many security features in place may fail to detect these malicious payload strings. To overcome this problem there are various methods described one such method is utilizing machine learning models to detect malicious strings by classifying the input strings given to the web applications. This paper describes the study of six binary classification methods Logistic regression, Naïve Bayes, SGD, ADABoost, Random Forrest, Decision trees using our own dataset and feature set.

***Keywords : Binary Classification, Machine Learning, Web Application*.**

I. INTRODUCTION

Web applications have become increasingly complex as they can perform multitude of tasks and have a lot of information processing capabilities. Latest developing techniques have made it very easy to implement a web application but due to these various underlying technologies that go into the development of web applications new security problems arise. There is an explosive growth of security incidents related to web applications according to statistics stated in [1] there are about ninety million attacks per week on web applications worldwide. Several agents play a role which make securing web applications highly strenuous, on the non-technical side of the things, delivery of the application to the users is one major factor that impedes security of the web application which leaves the web application open for attacks. Attackers often target web applications as they have an interface readily available, through which they could provide malicious input. This input would then get processed by a server and generate desired output that would be visible on the application. Various methods exist to detect and prevent attacks on web applications such as Secure Coding Practices, Web Application Firewalls (WAF), honeypots etc. Mod Security [2] is an open source and popular WAF which prevents attacks on web applications using rules that work using regular expressions and the HTTP traffic that flows is checked for attacks using these rules. Mod Security generally works with a default rule pack called the OWASP Core Rule Set [3] (OWASP CRS). Setting up this WAF and writing custom rules to stop advanced attacks is often a time taking as each rule must written manually based on the attack type and signature.

Revised Manuscript Received on August 05, 2019.

First Author Name*, Department of Cybersecurity Systems and Networks, Center for Cybersecurity Systems and Network, Amritapuri, India.

Dr. Rajesh Kannan Megalingam, Department of Electronics and Communication, Amrita Vishwa Vidyapeetham, Amritapuri, India.

II. BACKGROUND AND CONTRIBUTION

A. Previous Work

Successful attacks on web applications are often the direct result of exploiting the vulnerabilities present in the web application, inspecting and fixing these coding bugs may not be feasible always. We have already discussed about Mod Security a rule based WAF and how it must be manually configured to detect attacks. Another kind of WAF is [4] which is an anomaly based firewall which works utilizing an XML file to generate a path for every request to the web application which is checked with then compared with an existing normal behavior path if the path appears to be longer than the normal behavior path then the request is tagged as anomaly. Several other approaches exist to detect particular type of attack on the web application [7] discusses a proxy called SDriver that lies in between the web application and the web server to detect SQL injection based on signatures present in the proxy. A common mistake that programmers make is assuming that the user input would not be malicious [5] SQL injection [6] itself has a vast number of sub attack types like tautologies, union queries and piggy-backed queries [8] so detecting these along with attacks like XSS, Command Injection, etc. would be very difficult with manually configured WAF.

B. Problem Definition

The problem that we aim to solve is classifying the input given to a web application as malicious or regular.

C. Contribution

In this work we aim to identify the malicious input provided by the user utilizing supervised classification models to overcome the problem of manually updating the rules for WAF's or proxies and prevent attacks like XSS, SQL injection, Directory traversal, LDAP injection, Command Injection and File inclusion. To achieve this we designed our own feature set and dataset. Using the designed feature set and dataset we tested six classification techniques. Finally an evaluation

of these six classification techniques is given.

III. METHODOLOGY

Classifiers are supervised learning models that specialize in identifying data into either of the two classes defined. Here we have defined our classes as malicious and legal. All malicious strings are labelled as '1' and all legal strings are labelled as '0'. Data collection is a basic requirement for any machine learning algorithm.

A. Data Collection

In this phase we have attacked several Capture The Flag (CTF) [10] based web challenges to collect malicious input strings. Using fuzzing tools like sqlmap [11] to generate SQL injection strings, XSSer [12] to generate XSS strings, commix [13] to generate command injection strings. Collected the wordlists for directory traversal attacks. After collecting these strings we have labelled them accordingly with the attack type Table 1. SQL injection and XSS labels are shown in Fig 1 and Fig 2. The data set contains 100000 data items, 66000 are malicious strings and rest are valid legitimate strings.

Attack type	Label
Non-malicious string	Legal
File inclusion attack string	Inclusion
Directory Traversal	DIRB
LDAP Injection	LDAP
SQL Injection	SQL
XSS	XSS
Command Injection	CMD_Inj

Table 1 Attack type and label

First 5 lines of SQL

	payload	is_malicious	injection_type
0	'-'\n	1	SQL
1	' '\n	1	SQL
2	'&'\n	1	SQL
3	'^'\n	1	SQL
4	'*'\n	1	SQL

Fig 1 SQL Injection

First 5 lines of XSS

	payload	is_malicious	injection_type
0	<script>alert(123);</script>\n	1	XSS
1	<ScRipT>alert("XSS");</ScRipT>\n	1	XSS
2	<script>alert(123)</script>\n	1	XSS
3	<script>alert("hellox worldss");</script>\n	1	XSS
4	<script>alert("XSS")</script> \n	1	XSS

Fig 2 XSS

B. Feature Engineering

For converting the input data to numeric data a custom feature space was used containing 10 features such as SQL keywords present in the input string, JS keywords present in the input string, length of the input string etc. Table 2 describes the feature space.

Feature Name
SQL Keywords
Length
Non Printable characters
Punctuation characters
Min Byte
Max Byte
Mean Byte
Standard Deviation Byte
Distinct Bytes
Javascript keywords

Table 2 Features

Described below in Fig 3 how a string would get converted to the feature space.

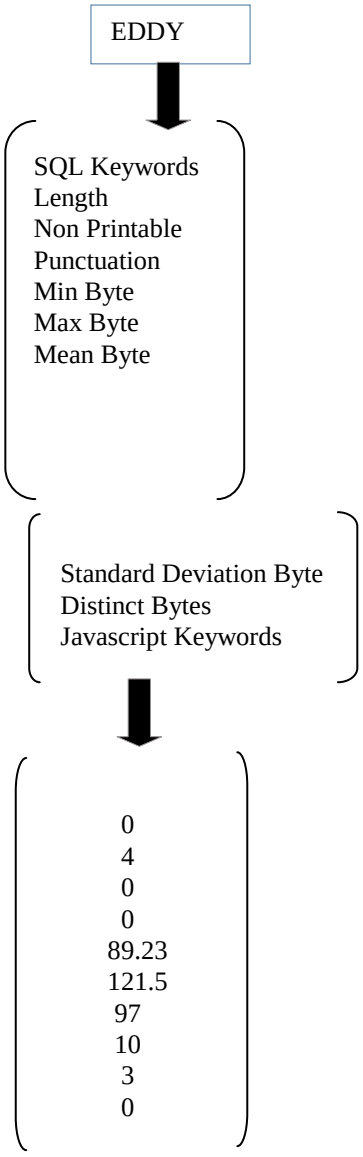

Fig 3 Feature Space

C. Feature Selection

Selecting the features that make a major impact for our dataset is very much necessary for training our models to achieve this we have performed a χ^2 scoring method to determine the best features Fig 4 shows the result. Length of the input and the punctuation marks in an input string had the most effect on the learning models for training.

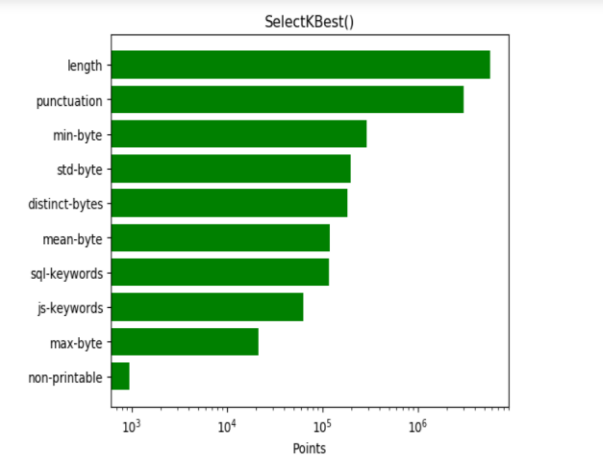

Fig 4 Best Features

IV. TRAINING, TESTING AND VALIDATION

When training and evaluating a machine learning model, a common practice is to split the data set into three separate parts; training set, validation set and a test set. The purpose of dividing the data set is to minimize the risk of curve-fitting. This phenomena occurs when the model is evaluated using the same data it has been trained on, which means that the model will be good at predicting already known data points, but the performance of predicting new unseen data points is unknown, and usually performing poorly. A second mistake that can result in a curve-fitted model is basing the selection of hyperparameters on the same data as the model is being optimized on hyperparameters are settings of the machine learning model that is algorithm-specific, for example the number of hidden layers in an ANN or the number of weak classifiers to use in AdaBoost. In summary: 1. Train a set of models with different hyperparameters on the training data 2. Evaluate the models on the validation data and pick the best model 3. The selected model can now have its true performance measured using the training data that is yet not seen by the model or the person/algorithm that selected the best model by the validation data. A popular technique commonly used when optimizing a model is to use cross-validation. The principle of this technique is to re-use the training and validation data by setting the validation data portion to an interval inside the old training data and resetting the validation data to training data. This can be done multiple times for better accuracy of the validation performance. In other words, it removes any eventual bias the validation data had by making a new portion of validation data for multiple iterations.

Performance metrics when evaluating models from the results of validation data or test data some performance metric has to be compared. The easiest is to sort the results by accuracy, e.g. the percent of successful classifications, but in cases where the distribution of classes among the data points is very skewed this metric might be deceptive. Other common performance metrics are calculated from the confusion matrix Table 3.

Abbrevation	Explanation
TP	True Positive Number of data points with class 1 also classified as 1
FP	False Positive Number of data points with class 0 but classified as 1
TN	True Negative Number of data points with class 0 also classified as 0
FN	False Negative Number of data points with class 1 but classified as 0

Table 3 Confusion matrix

The performance metrics used in addition to accuracy are: sensitivity, specificity, f1-score and AUC. The equation for sensitivity is shown in Equation 1 and can be interpreted as ”the percent of class 1 data points that were classified correctly”. [14]

$$Sensitivity = \frac{TP}{TP + FN} - (1)$$

Specificity is shown in Equation 2 and can be interpreted as ”the percent of class 0 data points that were classified correctly”. [14]

$$Specificity = \frac{TN}{TN + FP} - (2)$$

F1-score from Equation 3 is an all-around metric that is good to use instead of accuracy when the distribution of labels are skewed among the data set.

$$F1 - score = \frac{2 * precision * recall}{precison * recall} - (3) \quad where\ recall = sensitivity$$

$$precision = \frac{TP}{TP + FP}$$

Area under the curve, AUC, is a metric extracted from the Receiver operating characteristic (ROC). The ROC curve of a model is a plot with the true positive rate (sensitivity) as the y-axis and the false positive rate (1 − specificity) as the x-axis. The AUC metric is simply the percent of the graph that is under the ROC curve (0 < x-axis < 1 and 0 < y-axis < 1). [15]

V. RESULTS

The F1- Scores, Accuracy, Specificity, Sensitivity of all six of our classifiers is given in the below Fig 5. The top performing classifier was the random forest classifier with a F1-score of 0.997.

	F1-score	accuracy	sensitivity	specificity	auc	conf_matrix
custom RandomForest	0.997493	0.995717	0.980460	0.998336	0.998481	[[38396, 64], [129, 6473]]
custom DecisionTree	0.988897	0.981026	0.928961	0.989964	0.986863	[[38074, 386], [469, 6133]]
custom AdaBoostClassifier	0.988299	0.979983	0.919115	0.990432	0.994475	[[38092, 368], [534, 6068]]
custom Logistic	0.982894	0.970574	0.854287	0.990536	0.983940	[[38096, 364], [962, 5640]]
custom SGD	0.979669	0.965093	0.846865	0.985387	0.981120	[[37898, 562], [1011, 5591]]
custom MultinomialNB	0.970139	0.948893	0.810209	0.972699	0.971249	[[37410, 1050], [1253, 5349]]

Fig 5 Results

Lastly, the Receiver Operating Characteristic (ROC) curve is

drawn in Fig 6 for the top three and bottom three performing models. Recall from Table 4 which these models are. The ROC curve demonstrates two things. The first being a visualization of the area under the curve (AUC), which is so high for the top three models (exact values presented in Fig 6) that it is required to zoom in to see the boundaries in the upper left corner. The second thing visualized in the ROC curve is the sensitivity/specificity trade-off, which are the actual lines models are. The ROC curve demonstrates two things. The first being a visualization of the area under the curve (AUC), which is so high for the top three models (exact values presented in Fig 6 and Fig 7) that it is required to zoom in to see the boundaries in the upper left corner. The second thing visualized in the ROC curve is the sensitivity/specificity trade-off, which are the actual lines

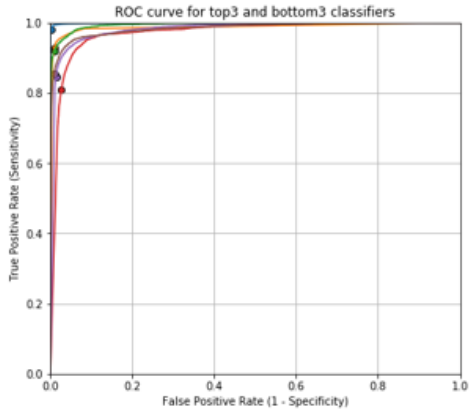

Fig 6 ROC Curve

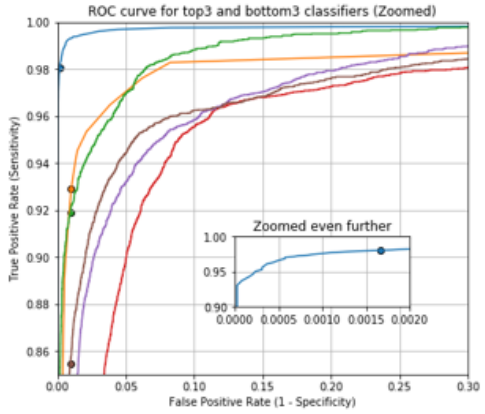

Fig 7 ROC Curve

VI. CONCLUSION

The main purpose of the paper is to understand the web application attacks and utilize the supervised learning algorithms to clearly detect the attacks caused by improper input provided to the web application. The classifier Random Forest has performed very well with an F1-score of 99.7 percent. A sub problem to solve within the scope of this project was to find relevant features for detecting malicious input. Ten custom features were found. Using a chi-squared based scoring function we were able to cherry pick the best features to make a classifiers as accurate as possible. The length of the input, the number of punctuation characters and the number of distinct bytes were the three best features that were found. The number of printable characters was however the worst one, and looking back at the distributions, this is not a surprise. Comparing different similar features we can notice some interesting differences. For example the minimum byte of a payload is far more deciding than maximum byte. The standard deviation and the number of distinct bytes also have far more impact than the mean.

REFERENCES

1. Web Attack Visualization | Akamai. (2019). Retrieved 18 July 2019, from https://www.akamai.com/us/en/resources/our-thinking/state-of-the-internet-report/web-attack-visualization.jsp
2. I. Trustwave Holdings, “Modsecurity: Open source web application firewall.” [Online]. Available: http://www.modsecurity.org/
3. OWASP. Owasp modsecurity core rule set project. [Online]. Available https://www.owasp.org/index.php/
4. Torrano-Gimenez, C., Perez-Villegas, A., & Alvarez, G. (2009). A self-learning anomaly-based web application firewall. In Computational Intelligence in Security for Information Systems(pp. 85-92). Springer, Berlin, Heidelberg.
5. Wassermann, G., & Su, Z. (2004, October). An analysis framework for security in web applications. In Proceedings of the FSE Workshop on Specification and Verification of component-Based Systems (SAVCBS 2004) (pp. 70-78).
6. Kc, G. S., Keromytis, A. D., & Prevelakis, V. (2003, October). Countering code-injection attacks with instruction-set randomization. In Proceedings of the 10th ACM conference on Computer and communications security (pp. 272-280). ACM.
7. Mitropoulos, D., & Spinellis, D. (2009). SDriver: Location-specific signatures prevent SQL injection attacks. computers & security, 28(3-4), 121-129.
8. Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In Proceedings of the IEEE International Symposium on Secure Software Engineering (Vol. 1, pp. 13-15). IEEE.
9. Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G. (2007, February). Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In NDSS (Vol. 2007, p. 12).
10. team, c. (2019). CTFtime.org / All about CTF (Capture The Flag). Retrieved 18 July 2019, from https://ctftime.org/about/
11. sqlmap: automatic SQL injection and database takeover tool. (2019). Retrieved 18 July 2019, from http://sqlmap.org/
12. XSSer: Cross Site "Scripter". (2019). Retrieved 18 July 2019, from https://xsser.03c8.net/
13. commixproject/commix.(2019). Retrieved 18 July 2019, from https://github.com/commixproject/commix
14. justmarkham/scikit-learn-videos. (2019). Retrieved 18 July 2019, from https://github.com/justmarkham/scikit-learn-videos/blob/master/09_classification_metrics.ipynb
15. Rogers, S., & Girolami, M. (2016). A first course in machine learning. Chapman and Hall, ch. Assessing Classification Performance, pp. 196–201.

AUTHORS PROFILE

Madduri Venkata Sai Soma Manish received his B.Tech degree in 2017 and is pursuing his Masters in Cyber Security Systems and Networking from Amrita Vishwa Vidyapeetham, Center for Cybersecurity Systems and Networks, Amritapuri, India.

Dr. Rajesh Kannan Megalingam is an electronics engineer leading research on humanitarian technologies with special emphasis on Robotics at HuT (Humanitarian Technology) Labs and Assistant Professor at ECE Department, Amrita Vishwa Vidyapeetham. He completed his undergraduate in Engineering from College of Engineering, Anna University, Chennai in 1997 and masters and PhD at Amrita Vishwa Vidyapeetham in 2010 and 2015 respectively.