Detection of Web Attacks using Ensemble Learning.

Ravi Pallam[1], Sai Prasad Konda[2], Lasya Manthripragada[3], Ram Akhilesh Noone[4]

[1]Assistant Professor, Dept. of Computer Science and Engineering, Anurag University, Hyderabad, India
[2,3,4]Student, Dept. of Computer Science and Engineering, Anurag University, Hyderabad, India

---***---

Abstract - *Web Applications are liable to information security threats due to the compelling information it acquires from the users. Possessing data is the most powerful thing in this day and age. Wrongly acquired data can be used to exploit a company- which can be devastating, both in financial and reputational damage. The threats according to OWASP Top 10 include SQLi, Cross-Site Scripting (XSS), XXE, etc., In this paper, we focus on building a tool- it uses ensemble learning algorithms to train the payloads (SQLi and XSS) and detect attacks when user gives input; if any such attacks are detected the public Ip address of the user will be blocked.*

Key Words: Web Security, Web application, Cross-Site Scripting, Vulnerabilities, SQL injection, Machine learning.

1. INTRODUCTION

The Open Web Application Security Project (OWASP) is a non-profit organisation dedicated to enhancing software security. The OWASP TOP 10 [1] is a regular knowledge document for web application protection and developers. It reflects widespread agreement on the most serious security threats to web applications.
The most pressing ones are SQLi and XSS attacks among the Top 10 threats. Every minute, Cybercrime costs $2,900,000 a minute, and top corporations pay $25 per minute for security breaches [2]. The need to reduce the risk for data breaches and attacks has been higher than ever. There is a necessity to build an application that can control and/or prevent unauthorized access to sensitive information.
Existing mechanisms to detect an attack are mostly static-based approaches and a very limited area of research is dedicated towards dynamic approaches using Machine learning [3] [4] [5].
We introduce the background of attacks in question i.e., SQLi and XSS in section 1.2 and 1.2, followed by a detailed explanation of different approaches in section 2. The proposed system and the evaluation of the results obtained from the said system are discussed in sections 3 and 4 respectively. And finally, a demonstration of our tool.

1.1 SQLi Attacks

Most of the websites have a system of authorizing existing users and registering new users. Every registered user has a set of email addresses/user names and passwords. These details are stored and managed in the server using a software system known as Relational DBMS. To access and manage the data stored in the RDBMS, a specific language called SQL is used. SQL stands for Structured Query Language. As defined, it uses queries structured in a specific order to interact with the RDBMS.
When a user logs in to a website, they submit a set of user names and passwords. These inputs are then checked with data stored in the database and if the user entered a valid user name and password, the user is authorized. A malicious user can gain unauthorized access to an account/website by entering malicious SQL statements instead of the user's name and password in a compromised web application. This is called SQL Injection (SQLi) [13][15].

There are three categories of SQLi:

1. Error-based SQLi:
   "1'UNION select 1,2,3--+" -> gives an error if the tables have a smaller number of columns than mentioned in the query.
2. Union-based SQLi:
   "1'UNION select database(), version()--+"-> name of the database and version are returned. Any information can be retrieved by mentioning it in the query.
3. Blind SQLi:
   a. Boolean: Attackers send a query which returns either TRUE or FALSE; this information can be used to infer what kind of data is stored.
   b. Time-based: Depending upon the time database takes to return a response, attackers designate whether the result is TRUE or FALSE.

1.2 XSS Attacks

Before JavaScript was released, websites were just a collection of static pages. The advent of JavaScript in the year 1995, brought forth a new level of interactivity of web pages. JavaScript helps us to build what we call dynamic web pages. It allows the developers to modify the web pages at the

client's side without the need for sending a request to the server.
A new case of a web application vulnerability known as Cross-Site Scripting (XSS) was discovered as JavaScript became more popular and was used more frequently. XSS is a security vulnerability that allows a malicious user to alter the code on a website such that when that website is opened by a user that code is executed in the user's browser.
“The malicious script is delivered to the user's browser via the web page or web application. Forums, message boards, and web pages with commenting capabilities are popular targets for Cross-site Scripting attacks” [14][16].
XSS attacks are classified into three types.

1. Reflected XSS: This can be as simple as writing a JavaScript code for an alert. Cookies and other vulnerable information can be obtained. It executes only once.
2. Stored XSS: Here the code written by the attacker is stored in the database and it is executed whenever that particular page is accessed.
3. DOM-based XSS: The attacker will inject malicious code in the DOM that will be reflected on the web page.

2. LITERATURE SURVEY

One of the most famous tools in detecting SQLi attacks is considered to be AMNESIA. “[…] presented AMNESIA, a prototype tool that implements our technique for Java-based web applications, and an empirical evaluation of the technique. The empirical evaluation of the technique was performed on a set of seven web applications that consisted of two applications developed by a student team, but also used by other researchers, and five real applications. AMNESIA was able to stop all of the 1,470 attacks that we performed on the considered applications without producing any false positive for the 3,500 legitimate accesses to the applications. Furthermore, AMNESIA proved to be quite efficient in practice, at least for the cases considered—the overhead imposed by the technique on the web application was barely measurable.” [6]

CANDID -“We have presented a novel technique to dynamically deduce the programmer intended structure of SQL queries and used it to effectively transform applications so that they guard themselves against SQL injection attacks. […] At a more abstract level, the idea of computing the symbolic query on sample inputs in order to deduce the intentions of the programmer seems a powerful idea that probably has more applications in systems security.” [7]

Buehrer et al. used a similar approach of comparing the generated queries with the ones that should have been generated (programmer intended) [17]. Other approaches include using random numerals or strings of characters for detection [21], SQL guard is also another where it uses parsing mechanisms [22]. Dynamic training is similar to pattern matching [23]. Another detection method is proposed by Fonseca et al, “this paper analyses 715 vulnerabilities and 121 exploits of 17 web applications using field data on past security fixes. Some web apps were written in a weakly typed language, while others were written in a strongly typed language. Results suggest that applications written with strong typed languages have a smaller number of reported vulnerabilities and exploits.” [30]

With the evolution of Machine Learning, Komiya et al proposed detecting both SQLi and XSS attacks using different classifier algorithms [5]. They implemented two different feature extraction techniques. One was blank separation and other was tokenization. Tokenization method was considered to be more efficient. To succeed [5], Sonali Mishra [3] did similar work but by using regular expressions. They inferred that naïve bayes [10] is not as efficient as gradient boosting algorithms. They concluded with significant difference in accuracies between gradient boosting and naïve bayes. The former having more accuracy.

However, very limited research has been done on XSS attacks. Komiya et al [5] implemented a Machine Learning algorithm on SQLi as well as XSS data. Because of the restricted availability of datasets, XSS has not been delved into much. Our paper discusses training XSS and SQLi data using ensemble learning techniques- GBM, Light GBM, XGBoost, and AdaBoost.

3. PROPOSED SYSTEMS

In this section, we introduce our proposed methodology. It will help detect malicious code by training the available data using machine learning algorithms. This approach was first proposed by Komiya et al [5]. We decided to implement and evaluate Naïve Bayes [27], SVM using a linear kernel [11][18][31] (results are not included since they’re already implemented) and Boosting [19][12][32][42] algorithms. For data cleaning we considered two approaches – tokenization (1) and also further cleaning the data by removing stop words, stemming (2)—they're discussed deeper in section 3.1.

3.1 Datasets

We trained machine learning algorithms using data obtained from Kaggle [8][9]. The data is first subjected to pre-processing. This will help in the classification algorithms producing better results.

As part of the pre-processing of data, it removes all the blank rows (If any) [26]. And it will change all the text to lower case. This is an essential step as python is case sensitive. The third most important step is the tokenization-each entry in the data will be broken into a set of words [24]. The data pre-processed until here is fit into the models, the accuracies were noted, which we consider (1). Further, we removed Stop words, non-numeric and performed word-stemming [25] (2).

After final processing (1) and (2), the end output is stored and is split into training and test data sets. The target variable is label encoded-this will transform classes into numerical values. Finally, we vectorized the words [28] using TF-IDF vectorizer. Now, we ran different algorithms to classify and check for accuracy. Fig-1 and Fig-2 show a step-wise pre-processing of the dataset for (1) and (2) techniques.

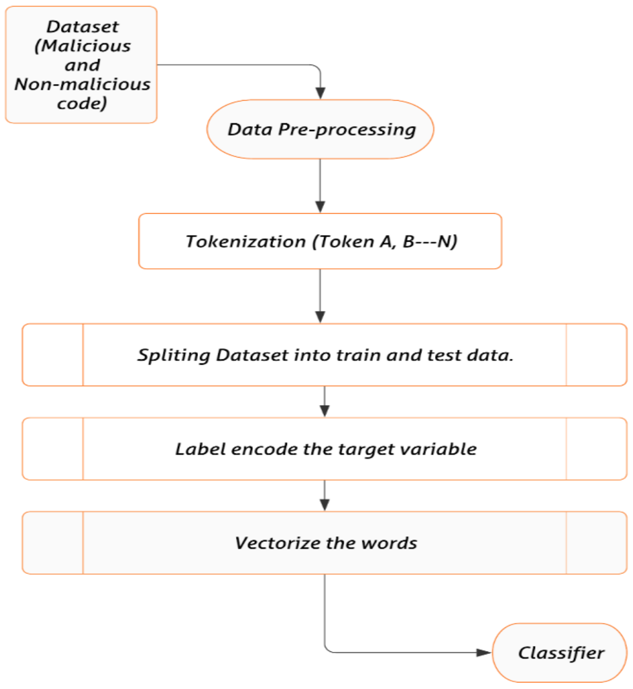

Dataset (Malicious and Non-malicious code)
Data Pre-processing
Tokenization (Token A, B---N)
Spliting Dataset into train and test data.
Label encode the target variable
Vectorize the words
Classifier

Figure-1 Flow of the proposed system (1)

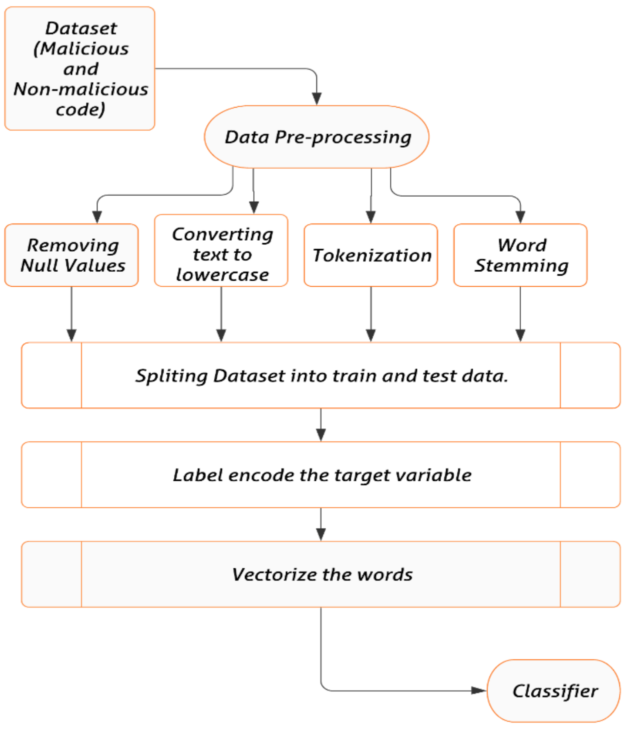

Dataset (Malicious and Non-malicious code)
Data Pre-processing
Removing Null Values
Converting text to lowercase
Tokenization
Word Stemming
Spliting Dataset into train and test data.
Label encode the target variable
Vectorize the words
Classifier

Figure-2 Flow of the proposed system (2)

4. RESULT ANALYSIS

We applied and assessed three ensemble learning approaches apart from Gradient Boosting- AdaBoost, XGBoost, and LightGBM. Gradient boosting algorithm has the high computational time. Adaboost suits well for a binary classification of data. LightGBM and XGboost are almost similar- very fast computational time and uses less memory as it uses histograms. LightGBM uses leaf-wise code splitting where as XGboost and the other algorithms use Level-wise node splitting. Two data-processing techniques are valuated and the results for each technique is shown in Tables, Table 1-4. Payloads available for SQLi are immensely larger than that of XSS payloads However, the XSS attacks datasets were more refined than SQLi attacks payload. The classification results of each machine learning approach are analysed by Precision, and Recall [20][29].

Figures from Figure 7 to Figure 16 are evaluation results-they're graphs plotted between precision and recall by (1). The first (1) data pre-processing technique and the (2) had a drastic difference in their computational time, which was expected as the data cleaning has cut down to one step i.e., tokenization. The former had higher accuracy than the latter. The in the latter technique, word stemming is performed which can be efficient only with data with no jargons. Removing stop words also can cause damage as query language has stop words as it's point of execution. Hence, (1) technique is considered to be more efficient, [5] and [1] also suggested the same technique. The accuracies are compared for the former and latter techniques in the figures- Fig- 7 and Fig-8. These evaluation results lead us to believe that (1) process can yield higher accuracies. Establishing that, Light GBM algorithm has shown a whooping accuracy of 99.5%. Since it is faster and utilizes lower memory, it implied that it was the best model to use.

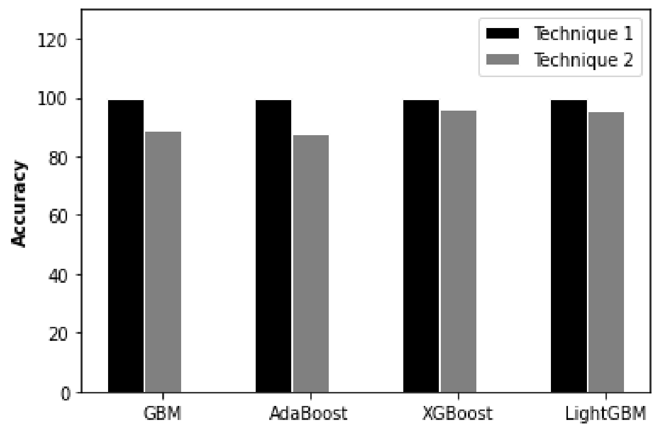


Figure 3- Accuracies by (1) and (2) on SQLi attacks

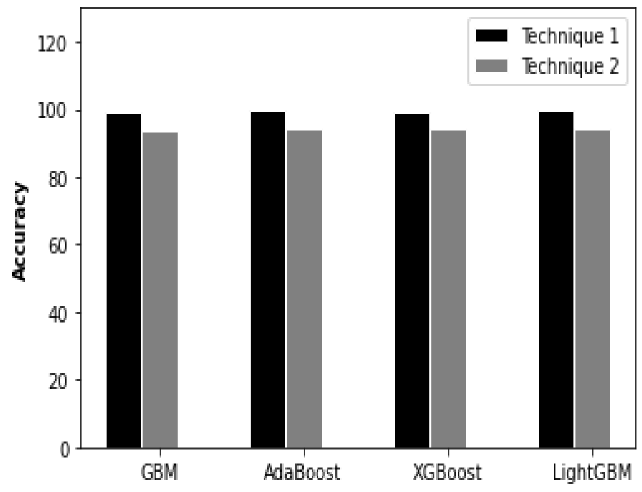


Figure 4- Accuracies by (1) and (2) on XSS attacks

	Accuracy	Avg. Precision	Recall
Gradient Boosting	99.23%	0.99	0.99
AdaBoost	99.50%	0.99	0.99
XGBoost	99.39%	0.99	0.99
Light GBM	99.51%	0.99	0.99

Table-1 SQLi attacks (1)

	Accuracy	Avg. Precision	Recall
Gradient Boosting	93.71%	0.89	0.92
AdaBoost	94.45%	0.89	0.92
XGBoost	94.33%	0.89	0.92
Light GBM	94.22%	0.89	0.92

Table-2 SQLi attacks (2)

	Accuracy	Avg. Precision	Recall
Gradient Boosting	99.50%	1.0	1.0
AdaBoost	99.56%	1.0	1.0
XGBoost	99.54%	1.0	1.0
Light GBM	99.59%	1.0	1.0

Table-3 XSS attacks (1)

	Accuracy	Avg. Precision	Recall
Gradient Boosting	89.25%	0.90	0.89
AdaBoost	87.78%	0.89	0.89
XGBoost	96.01%	0.96	0.96
Light GBM	95.54%	0.96	0.96

Table-4 XSS attacks (2)

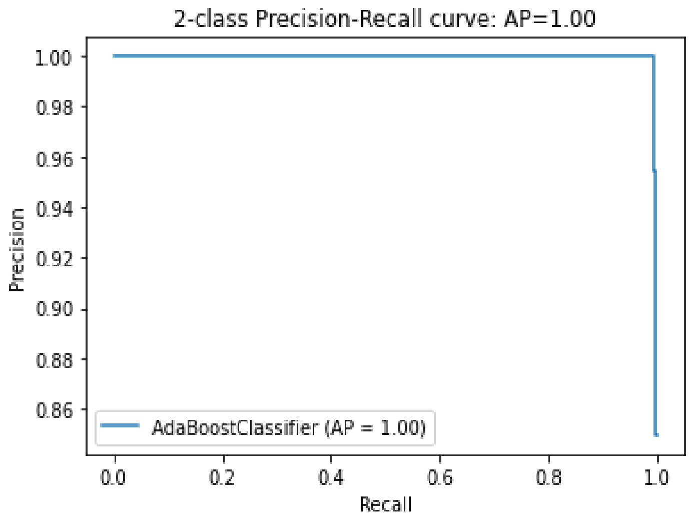


Figure-9 The Recall-Precision graph of Adaboost in XSS

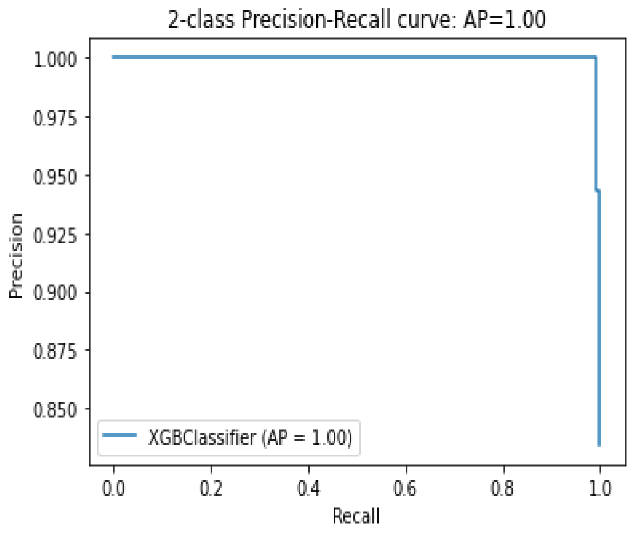


Figure-10 The Recall-Precision graph of XGBoost in XSS

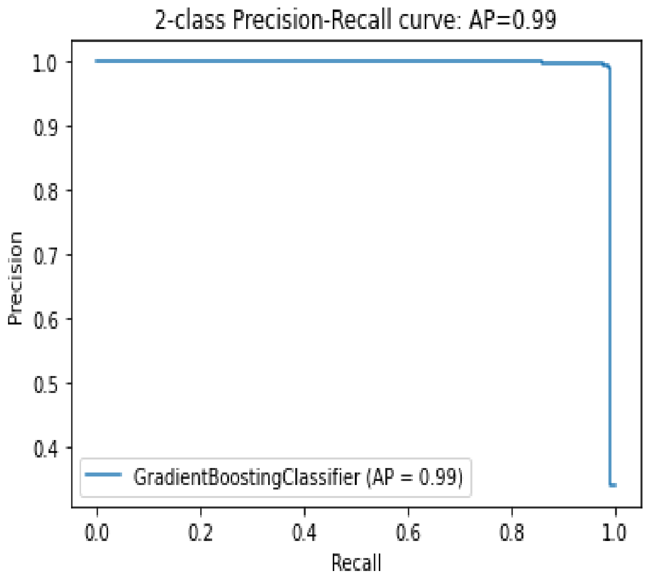


Figure-11 The Recall-Precision graph of Gradient boosting in XSS

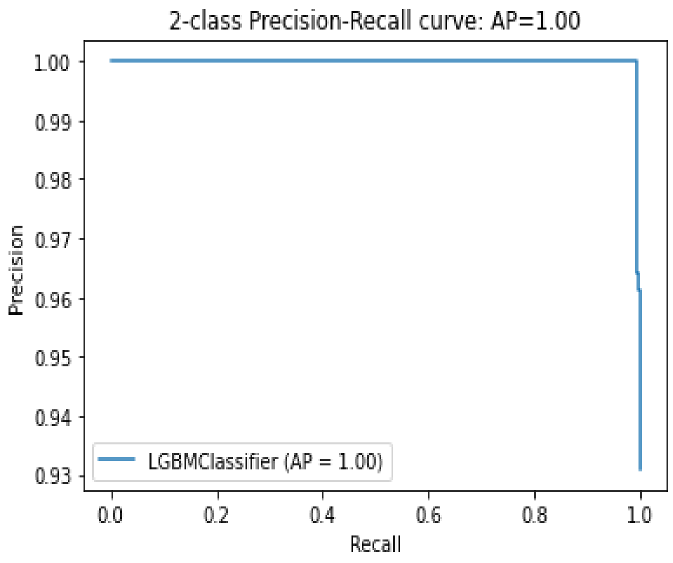


Figure-12 The Recall-Precision graph of Light GBM in XSS

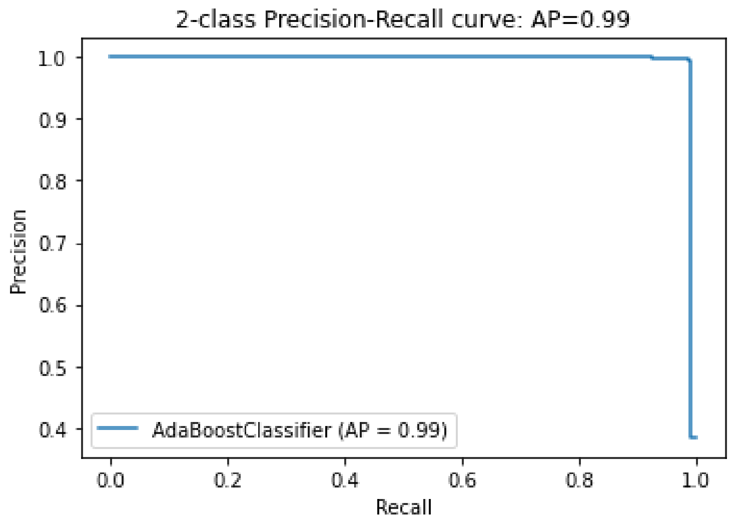


Figure-13 The Recall-Precision graph of Adaboost in SQLi

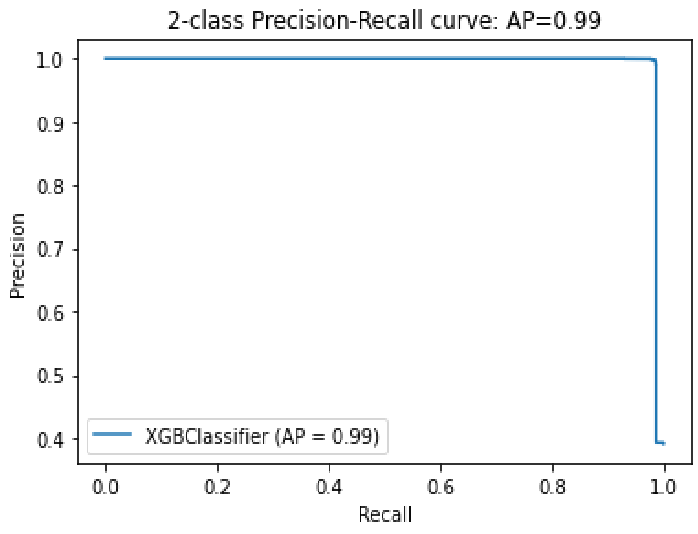


Figure-14 The Recall-Precision graph of XGBoost in SQLi

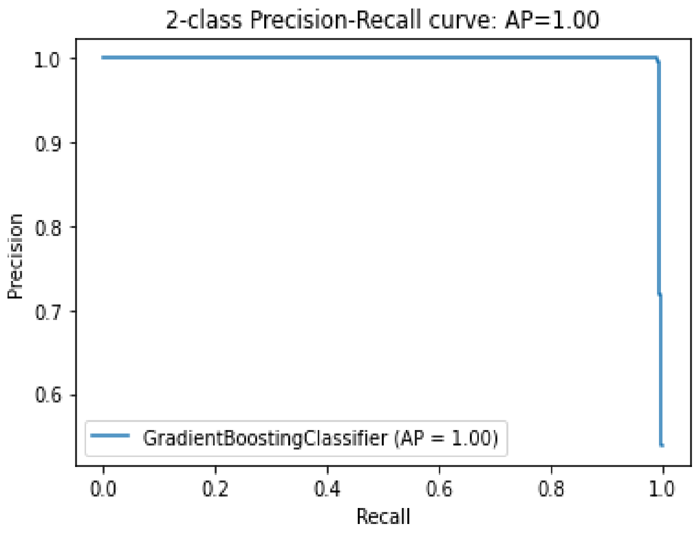


Figure -15 The Recall-Precision graph of Gradient boosting in SQLi

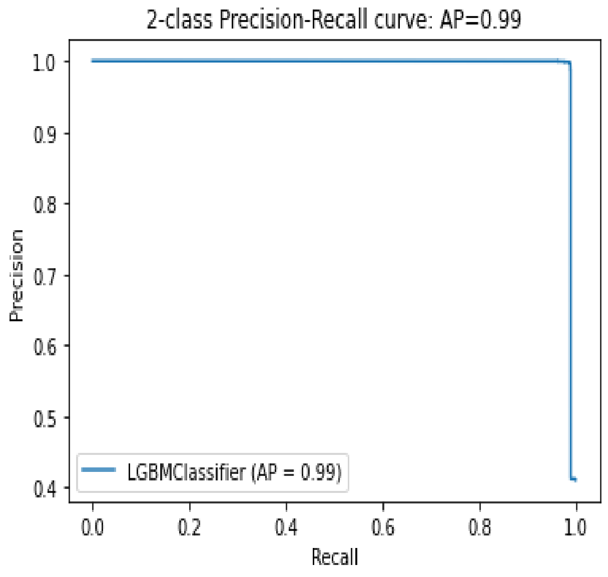
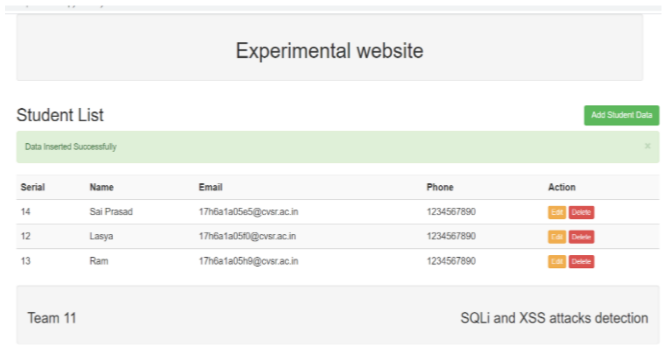


Figure -16 The Recall-Precision graph of Light GBM in SQLi

5. IMPLMENTATION

After developing the model to detect and prevent SQLi and XSS attacks, we deployed the model onto a CRUD application developed using flask [33] and the database we utilized was MySQL [34]. Whenever a user provides dynamic input, the model sanitizes the input to determine whether or not malicious code is there before committing to the database. If the model detects any suspicious activity, it will prompt an error saying there has been an attempt to breach. Figures, Fig-17-Fig 20 display a sample demonstration of how our experimental website works when a malicious code is given as input. The data for the models used to deploy in this specific website was pre-processed by (1) technique since it showed significant results.

When any attack is detected instead of just giving a warning to the client or in this case an attacker, we built a tool which will sanitize the input using technique (1) before passing it through the database, but if any malicious activity is found, the public IP address of that particular user is stored. Whenever said user tries to access the website again at a later time, our tool will find that the public IP address is already in the ban list and will not grant access to the user. Figure-20 gives a simple demonstration of the tool. The machine learning algorithm with which the pre-processed data is predicting is Light GBM as it gave significant accuracies, when compared with other boosting algorithms. (Table 1 and Table 3)


Experimental website

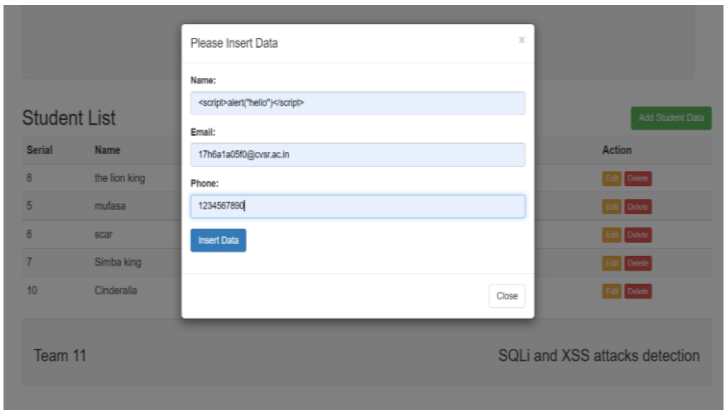
Student List

Data Inserted Successfully

Serial	Name	Email	Phone	Action
14	Sai Prasad	17h6a1a05e5@cvsr.ac.in	1234567890	Edit Delete
12	Lasya	17h6a1a05f0@cvsr.ac.in	1234567890	Edit Delete
13	Ram	17h6a1a05h9@cvsr.ac.in	1234567890	Edit Delete

Team 11 SQLi and XSS attacks detection

Figure-17 A Website built to demonstrate working of SQLi and XSS models


Please Insert Data

Name: <script>alert("hello")</script>

Email: 17h6a1a05f0@cvsr.ac.in

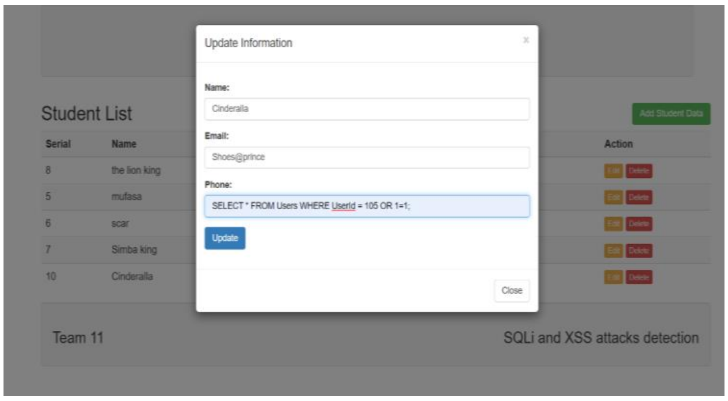
Phone: 1234567890

Student List

Serial	Name	Action
8	the lion king	Edit Delete
5	mufasa	Edit Delete
6	scar	Edit Delete
7	Simba king	Edit Delete
10	Cinderalla	Edit Delete

Team 11 SQLi and XSS attacks detection

Figure-18 Deliberately giving an SQL query to prompt an attack


Update Information

Name: Cinderalla

Email: Shoes@prince

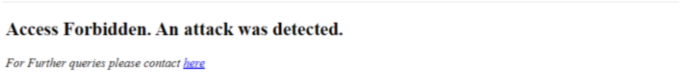
Phone: SELECT * FROM Users WHERE UserId = 105 OR 1=1;

Student List

Serial	Name	Action
8	the lion king	Edit Delete
5	mufasa	Edit Delete
6	scar	Edit Delete
7	Simba king	Edit Delete
10	Cinderalla	Edit Delete

Team 11 SQLi and XSS attacks detection

Figure-19 Deliberately giving malicious code to prompt XSS attack


Access Forbidden. An attack was detected.

For Further queries please contact here

Figure-20 Denying access if there is an attack

6. CONCLUSION

Web applications are exploited on a day-to-day basis as they contain sensitive information. Two of the most common exploitation techniques are SQLi and XSS. Our project attempts to solve these perpetual attacks by building classifiers on payloads of both SQLi and XSS. Gathering from our analysis, tokenization of data is the best technique for pre-processing (1), and also the computational time was significantly lesser for (1). After Evaluating the results, Light GBM clearly stood out from the rest of the boosting algorithms with 99.51% and 99.59% accuracies for SQLi and XSS respectively. For further enhancement- investigation on malicious code i.e., curating a more efficient and specific dataset is recommended as the payloads are very limited.

7. REFERENCES

[1] The Open Web Application Security Project (OWASP). The Ten Most Critical Web Application Security Risks 2017. https://owasp.org/www-project-top-ten/2017/Top_10.html

[2] The Evil Internet Minute 2019, RiskIQ. https://www.riskiq.com/resources/infographic/evil-internet-minute-2019/

[3] Mishra, Sonali, "SQL Injection Detection Using Machine Learning" (2019). *Master's Projects*. 727.DOI: https://doi.org/10.31979/etd.j5dj-ngvb https://scholarworks.sjsu.edu/etd_projects/727

[4] Tajpour, Atefeh & Ibrahim, Suhaimi & Masrom, Maslin. (2011). SQL Injection Detection and Prevention Techniques. International Journal of Advancements in Computing Technology. 3. 82-91. 10.4156/ijact.vol3.issue7.11.

[5] R. Komiya, I. Paik and M. Hisada, "Classification of malicious web code by machine learning," 2011 3rd International Conference on Awareness Science and Technology (iCAST), 2011, pp. 406-411, doi: 10.1109/ICAwST.2011.6163109.

[6] William G. J. Halfond , Alessandro Orso, "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks", Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, November 07-11, 2005.

[7] Prithvi Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks", ACM Transactions on Information and System Security (TISSEC), v.13 n.2, p.1-39, February 2010.

[8] Kaggle SQL injection dataset-- https://www.kaggle.com/syedsaqlainhussain/sql-injection-dataset?select=sqli.csv

[9] Kaggle XSS dataset for deep learning-- https://www.kaggle.com/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning

[10] A. Joshi and V. Geetha, "SQL Injection detection using machine learning," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kanyakumari, 2014, pp. 1111-1115.

[11] Understanding Support Vector Machine(SVM) algorithm from examples-- https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[12] Chapter 12- Gradient Boosting-- https://bradleyboehmke.github.io/HOML/gbm.html

[13] Jianwei Hu, Wei Zhao, and Yanpeng Cui. 2020. A Survey on SQL Injection Attacks, Detection and Prevention. In <i>Proceedings of the 2020 12th International Conference on Machine Learning and Computing</i> (<i>ICMLC 2020</i>). Association for Computing Machinery, New York, NY, USA, 483–488. DOI:https://doi.org/10.1145/3383972.3384028

[14] Garcia-Alfaro, Joaquin & Navarro-Arribas, Guillermo. (2009). A Survey on Cross-Site Scripting Attacks.

[15] J. Abirami, R. Devakunchari and C. Valliyammai, "A top web security vulnerability SQL injection attack — Survey," 2015 Seventh International Conference on Advanced Computing (ICoAC), Chennai, 2015, pp. 1-9.

[16] S. K. Mahmoud, M. Alfonse, M. I. Roushdy and A. M. Salem, "A comparative analysis of Cross Site Scripting (XSS) detecting and defensive techniques," 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS), 2017, pp. 36-42, doi: 10.1109/INTELCIS.2017.8260024.

[17] Gregory Buehrer , Bruce W. Weide , Paolo A. G. Sivilotti, "Using parse tree validation to prevent SQL injection attacks", Proceedings of the 5th international workshop on Software engineering and middleware, September 05-06, 2005

[18] N.Cristianini and J.Shawe-Taylor, "An Introduction to Support Vector Machine and Other Kernel-based Learning Methods," Cambridge University Press, 2000.

[19] Natekin, Alexey & Knoll, Alois. (2013). Gradient Boosting Machines, A Tutorial. Frontiers in neurorobotics. 7. 21. 10.3389/fnbot.2013.00021.

[20] David L.Olson and D.Delen, "Advanced Data Mining Techniques," Springer; I edition, pp. 138, Feb 2008.

[21] S.W.Boyd and AD.Keromytis, "SQLrand: Preventing SQL Injection Attacks," Proc. the 2nd Applied Cryptography and Network Security (ACNS) Conference, pp. 292-302, Jun 2004.

[22] G.T.Buehrer, RW.Weide, and P.AG.Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," International Workshop on Software Engineering and Middleware (SEM), 2005.

[23] V.Haldar, D.Chandra, and M.Franz, "Dynamic Taint Propagation for Java," Proc. 21 st Annual Computer Security Applications Conference, Dec 2005.

[24] Rai A., Borah S. (2021) Study of Various Methods for Tokenization. In: Mandal J., Mukhopadhyay S., Roy A. (eds) Applications of Internet of Things. Lecture Notes in Networks and Systems, vol 137. Springer, Singapore. https://doi.org/10.1007/978-981-15-6198-6_18.

[25] Pradana, Aditya & Hayati, Mardhiya. (2019). The Effect of Stemming and Removal of Stopwords on the Accuracy of Sentiment Analysis on Indonesian-language Texts.

Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control. 4. 10.22219/kinetik.v4i4.912.

[26] Imputation of missing values- https://scikit-learn.org/stable/modules/impute.html

[27] Kaviani, Pouria & Dhotre, Sunita. (2017). Short Survey on Naive Bayes Algorithm. International Journal of Advance Research in Computer Science and Management. 04.

[28] Kumari, Anita & Shashi, M.. (2019). Vectorization of Text Documents for Identifying Unifiable News Articles. International Journal of Advanced Computer Science and Applications. 10. 305. 10.14569/IJACSA.2019.0100742.

[29] Goutte, Cyril & Gaussier, Eric. (2005). A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. Lecture Notes in Computer Science. 3408. 345-359. 10.1007/978-3-540-31865-1_25.

[30] J. Fonseca, N. Seixas, M. Vieira and H. Madeira, "Analysis of Field Data on Web Security Vulnerabilities," in IEEE Transactions on Dependable and Secure Computing, vol. 11, no. 2, pp. 89-100, March-April 2014, doi: 10.1109/TDSC.2013.37.

[31] Sklearn- SVM classifiers- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[32] Sklearn- gradient boosting- https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[33] Flask web development, one drop at a time- https://flask.palletsprojects.com/en/2.0.x/

[34] MySQL, PHPMyAdmin- https://www.phpmyadmin.net/docs/

[35] F. Huang, G. Xie and R. Xiao, "Research on Ensemble Learning," 2009 International Conference on Artificial Intelligence and Computational Intelligence, 2009, pp. 249-252, doi: 10.1109/AICI.2009.235.