Prediction of SQL Injection Attacks in Web Applications

Chamundeswari Arumugam[1(✉)], Varsha Bhargavi Dwarakanathan[1(✉)], S. Gnanamary[1(✉)], Vishalraj Natarajan Neyveli[1(✉)], Rohit Kanakuppaliyalil Ramesh[1(✉)], Yeshwanthraa Kandhavel[1(✉)], and Sadhanandhan Balakrishnan[2(✉)]

[1] Department of Computer Science and Engineering, SSN College of Engineering, Kalavakkam, Chennai, Tamil Nadu, India
chamundeswaria@ssn.edu.in, {varshal5120, gnanamary15304, vishalrajl612, rohitl6086, yeshwanthraal6128}@cse.ssn.edu.in

[2] Indium Software (India) Limited, Chennai, India
Sadhanandh.b@indiumsoft.com

Abstract. As web applications become increasingly complex and connected, it becomes imperative to reduce the vulnerabilities in applications. SQLIA is a part of OWASP vulnerabilities and it is extremely important to prevent them. The proposed system aims to predict the occurrence of SQLIA on a given server, with applications deployed on it, from a given source, at a particular time. This prediction can be done with the help of JMeter tool. Apache JMeter is used to simulate logs data. From this, one can pre-process, extract features, and classify, which is then fed to a model for prediction of SQLIA.

Keywords: SQL injection · Web application · Classification · Prediction

1 Introduction

It is no secret that the world has become increasingly dependent on technology in the past decade. Many factors have led an increasing number of organizations and individuals to rely on web-based applications to provide access to a variety of services. However, insecure software is undermining our security-critical environments such as finance, healthcare, defense, energy, etc. As applications become increasingly complex and connected, the importance of achieving application security increases exponentially. It is imperative to reduce the vulnerabilities in applications and make them impervious to attacks.

Web application vulnerabilities [15] involve a system flaw or weakness in a web-based application. They have been around for years, largely due to not validating or sanitizing form inputs, misconfigured web servers, and application design flaws. Validation checks if the input meets a set of criteria (such as a string contains no standalone single quotation marks). Sanitization [7] modifies the input to ensure that it is valid (such as doubling single quotes).

S. Misra et al. (Eds.): ICCSA 2019, LNCS 11622, pp. 496–505, 2019.
https://doi.org/10.1007/978-3-030-24305-0_37

Many of the servers that store critical data for websites and services use SQL to manage the data in their databases. An SQL Injection Attack (SQLIA) [16] specifically targets this kind of server, using malicious code to get the server to divulge information it normally would not. Successful SQLIA typically occur because a vulnerable application does not properly sanitize inputs provided by the user. Cross Site Scripting (XSS) [8] attack also involves injecting malicious code into a website. Cross-site scripting allows an attacker to execute malicious scripts in another user's browser. One of the most common ways an attacker can deploy a cross-site scripting attack is by injecting malicious code into an input field that would be automatically run when other visitors view the infected page.

The objective of the paper is to predict SQLIA in the web application. Apache JMeter, an open source software was used as a load generator to create log data. From log data, the preprocessing is done to perform feature extraction. With respect to the prediction, the logistic regression model was used. The paper is organized as follows: Sect. 2 describes about the related work of SQLIA while Sect. 3 describes about the proposed system and implementation of this work. Section 4 discuss about the results while Sect. 5 describes about the conclusion.

2 Literature Survey

Scholte et al. [1] present a study of input validation vulnerabilities with the aim of gaining deeper in-sights into how these common web vulnerabilities can be prevented. They focus on the relationship between the specific programming language used to develop web applications and the vulnerabilities that are commonly reported, and found that most SQLIA and XSS vulnerabilities can be prevented using straightforward validation mechanisms based on common data types. Alkhalaf et al. [2] talk about input validation and sanitization. They propose to use the validation and sanitization functions as input to the algorithm, which will perform differential repair. The main aim is to remove redundancy at server side as well as at other checking instances. The repair algorithm takes the validation and sanitization functions as input and aims to repair the semantic difference.

Frajták et al. [3] concentrate on reducing user input validation code in web applications using Pex extension. Pex is a white box testing input generator for .NET applications. This approach reduces the amount of code created by developers. The code that validates the values of method input parameters does not have to be duplicated in JavaScript and this code is updated whenever a change is made to the code of the method that handles client request. Li et al. [4] summarizes all the known vulnerabilities and attacks. They present a survey of recent techniques and approaches for server side securing of web applications. Cho et al. [5] proposed a technique, which verifies input values of Java-based web applications using static byte code instrumentation and runtime input validation. This approach searches for target methods or object constructors in compiled Java class files, and statically inserts byte code modules.

Medeiros et al. [6] discussed the use of static analysis to detect vulnerabilities in web applications. They then use the output and apply data mining techniques to detect and reduce the number of false positives. Solomon et al. [9] applied machine learning

predictive analytics to predict and prevent SQLIA in cloud hosted web application. An web proxy application programming interface to accurately predict malicious SQLIA in web request was provided as a web service for protecting back-end database. Jingling et al. [14] proposed a dynamic taint tracking approach to explore SQLI and XSS vulnerabilities in web applications. WOVSQLI [10], one of the SQLIA tool was developed to detect the attack. This tool with SQL word vector and LSTM neural networks was used to detect the SQLIA in web application. A large dataset was used to demonstrate the accuracy of the tool.

A tool to detect SQLIA AMNESIA [12] was implemented for Java-based web applications and tested on five real time application without producing false positive. Haldar et al. [11] proposed a tagging and tracking approach to detect the user input in web application so as to prevent number of attacks. Komiya et al. [13] suggested to adapt machine learning approach for classification and detection of malicious user inputs in web code. Thus, vulnerability prediction is important and this work considers the SQLIA for prediction.

3 Proposed System

For a given system, where a system is considered to be a server and the applications hosted on it, the solution aims to predict the chance of SQLIA to occur, from a particular place, at a particular time. For prediction, raw data is collected from the server logs. The Source IP gives the location from which a particular request is generated. The data and time give information as to when a particular request hits the server. This data from the server's log is pre-processed and the dataset is formed. Therefore, given a server and deployed applications, there is historic data. As the server gets more and more hits, the log data also increases, thus enabling the model to learn better, and in turn predict more accurately in real time.

From an application point-of-view, given a query to an application, it has to go via the server. This again gets recorded in the logs. Now, using request parameters and destination page information, it can also be found out as to what particular application this attack was directed at. The reason this is done is as follows. Consider Application A, as a standalone application may have some vulnerability based on the code. However, A can be deployed on Server X as well as on Server Y. Request to different servers will be different, in the sense that, they can come from different sources, and at different times, and A might be less vulnerable when deployed on X than it might be when deployed on Y. To tackle this problem, the proposed solution represented in Fig. 1, considers both the server and the applications hosted on it at any time as a whole system and then predicts SQLIA. The data saved in the access log is preprocessed and features are extracted. This is followed by classification and prediction.

3.1 Implementation

SQL (Structured Query Language) injection [16] is an important attack methodology that targets the data residing in a database through the firewall that shields it. The attack takes advantage of poor input validation in code and website administration. SQL

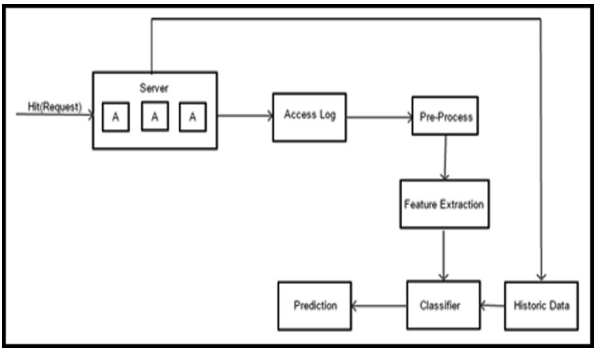

Fig. 1. Proposed system

Injection Attacks occur when an attacker is able to insert a series of SQL statements into a query by manipulating user input data in to a web-based application. The attacker can take advantage of web application programming security flaws, pass unexpected malicious SQL statements, and query through a web application for execution by the back-end database.

For example, consider the PHP code segment-1 as provided in Fig. 2. If the user enters value; DROP TABLE table; as the input, the query becomes as shown in Fig. 3. which is undesirable, as here the user input is directly compiled along with the pre-written SQL query. Hence the user will be able to enter an SQL query required to manipulate the database.

```
$variable = $_POST['input'];
mysql_query("INSERT INTO `table` (`column`) VALUES ('$variable')");
```

Fig. 2. PHP code segment-1

```
INSERT INTO `table` (`column`) VALUES('value'); DROP TABLE table;--')
```

Fig. 3. PHP code segment-2

3.2 Creation of Dataset and Classification

Apache JMeter is an Apache project that can be used as a load testing tool for analyzing and measuring the performance of a variety of services, with a focus on web applications. The Apache JMeter application is a open source software, a Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

Apache JMeter may be used to test performance both on static and dynamic resources, Web dynamic applications. It can be used to simulate a heavy load on server, group of servers, network or object to test its strength or to analyze overall performance under different load types. Here, JMeter is used to simulate queries to the server and generate log data as shown in Fig. 4.

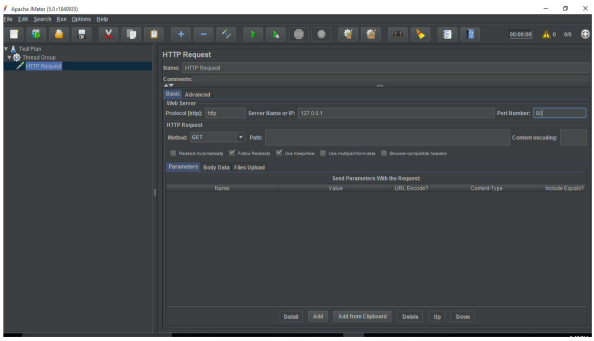

Fig. 4. JMeter tool to generate log data

Any request to the web server is recorded in its logs. This log data contains SourceIP, Date and Time, Request Parameter, and Destination pages. This raw data is used for creation of the dataset. For creation of log data, two PHP web applications have been used as shown in Fig. 5.

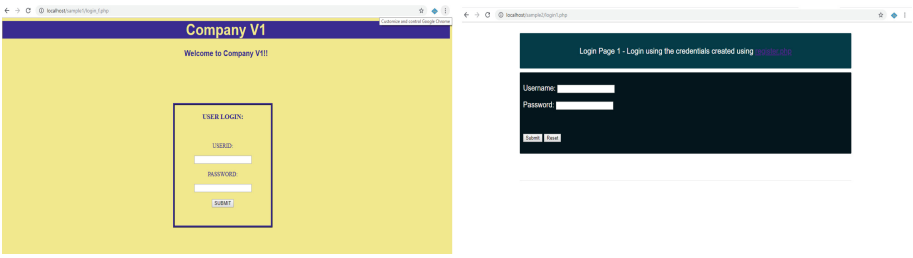

Fig. 5. Two PHP web application to create log data

WAMP server has been used for deployment of these applications. Both applications work on MySQL Database. Here, in order to create a huge training set, log data is simulated manually. These applications were hit with numerous requests via JMeter, in order to create multiple entries in the access log file. Working of JMeter is represented in Fig. 6.

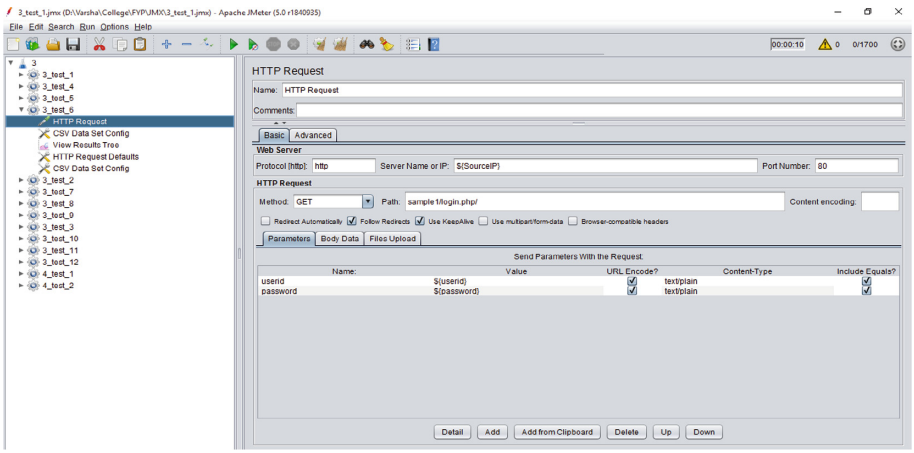

Fig. 6. JMeter working

Requests to these applications are recorded in the Apache Access logs. The access log is stored as a text file. This text file is converted into a csv file. From csv file, the *Source IP, Date, Time, Request parameter* and *Target application* features are extracted it is represented in Fig. 7. SQLIA related key words are extracted from the Request Parameter in order to predict SQLIA.

SourceIP	Time	Request	Sqlia
192.168.0.104	[27/Feb/2019:15:55:33	/sample1/login.php/?userid=Drop+table&password=pass	
192.168.0.103	[27/Feb/2019:15:55:33	/sample1/login.php/?userid=user1&password=pass1	
192.168.0.103	[27/Feb/2019:15:55:33	/sample1/login.php/?userid=user2&password=pass2	
192.168.0.106	[27/Feb/2019:15:55:33	/sample2/?products=-+%27+order+by+5+--+%2F%2F	
192.168.0.104	[27/Feb/2019:15:55:33	/sample1/login.php/?userid=Drop+table&password=pass	
192.168.0.106	[27/Feb/2019:15:55:33	/sample1/login.php/?userid=user8&password=pass8	
192.168.0.106	[27/Feb/2019:15:55:33	/sample2/?products=pillows	
192.168.0.103	[27/Feb/2019:15:55:33	/sample2/?products=-+%27+union+select+1	
192.168.0.106	[27/Feb/2019:15:55:33	/sample2/?products=book+shelf++	
192.168.0.105	[27/Feb/2019:15:55:33	/sample2/login1/?username=user2&password=pass2	

Fig. 7. Feature extraction from access log.

85 SQLIA commands where collected from research and they are used to compare with the SQL command in *Request parameter*. Using this the query is classified as *SQLIA* or not. These data is converted into a Dataset that can be fed into a logistic regression model. The Dataset comprises of features *Source IP, Time, Target application* and *SQLIA* and it is represented in Fig. 8.

SourceIP	Time	app	Sqlia1
2	4	1	1
1	4	1	0
1	4	1	0
4	4	2	1
2	4	1	1
4	4	1	0
4	4	2	0
1	4	2	0
4	4	2	0
3	4	3	0
3	4	3	1
2	4	3	0
4	4	3	1

Fig. 8. Data set for prediction

4 Prediction

Data Prediction refers to an area of statistics that deals with extracting information from data and using it to predict trends and behavior patterns. Here, it is done to predict the chance of SQLIA to occur on a system (Server and applications deployed on it) from a given place, at a particular time. For this prediction, logistic regression is used.

Logistic regression [17] is an classification algorithm, that is used where the response variable is categorical. The idea of logistic regression is to find a relationship between features and probability of particular outcome. Logistic regression works with binary data, where either the event happens (1) or the event does not happen (0). So given some feature it tries to find out whether some event y happens or not. So *y* can be either 0 or 1. In the case where the event happens, *y* is given the value 1. If the event does not happen, then *y* is given the value of 0. Logistic regression uses the sigmoid function, which gives an 'S' shaped curve to model the data. The curve is restricted between 0 and 1, so it is easy to apply when *y* is binary.

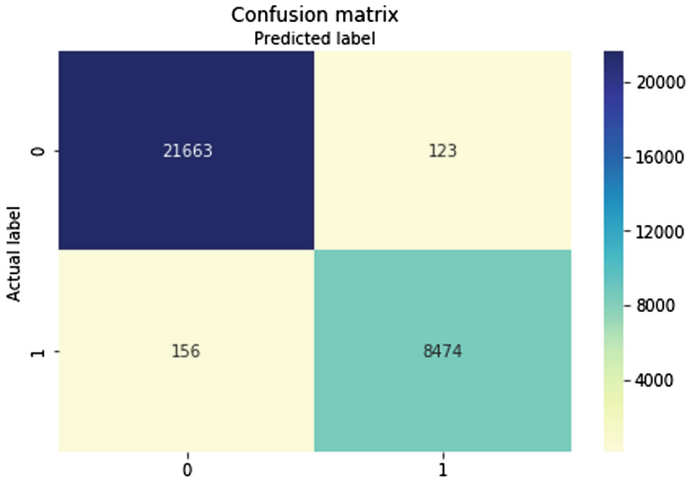

Fig. 9. Confusion matrix

From research, it was found that the following words occur with decreasing order of frequency in SQLIA Queries: Union Select, All, And, where, or, as, from, like, etc. Thus, the model was built and trained according to the key-words found in the request parameter. Out of a data set of about 1,00,000, 70% was used for training, and the rest for testing. Dataset comprising of *Source IP, Time, Target application* and *SQLIA* are fed to the model. For logistic regression, consider *Source IP, Time, Target application*, and *SQLIA* as independent variables and SQLIA as dependent variables. This model gives a 72% accuracy. This can be seen in the confusion matrix and the ROC curve as shown in Figs. 9 and 10.

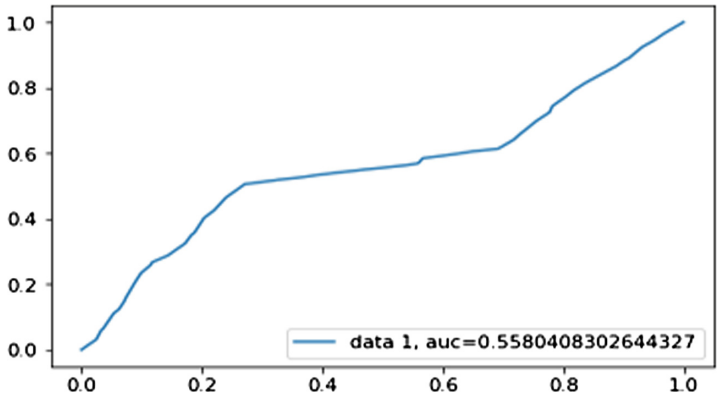

Fig. 10. ROC curve

5 Conclusion

Vulnerabilities prediction is a challenging task and it is addressed in this work. Here the web application was developed and deployed in a server. For the deployed application, the JMeter software was used to generate the log files. The log files was used here and SQL queries statement was taken for prediction. The chance of SQLIA to occur on a deployed web application from a given place, at a particular time was predicted. For this prediction, logistic regression was used on 1,00,000 data points. 70% was used for training, and the rest for testing. The model gives an 72% accuracy to predict SQL injection.

As a future work, a module can be implemented to raise an alarm, when an SQL Injection Attack occurs. Also, from the Source IP, one can backtrack to determine the actual physical location from which the attack is coming. The actual user can also be identified in the process. Deep learning methods and Tree regressors can be implemented for better results.

References

1. Scholte, T., Robertson, W., Balzarotti, D., Kirda., E.: An empirical analysis of input validation mechanisms in web applications and languages. In: 27th Annual ACM Symposium on Applied Computing, pp. 1419–1426 (2012)
2. Alkhalaf, M., Aydin, A., Bultan, T.: Semantic differential repair for input validation and sanitization. In: ACM International Symposium on Software Testing and Analysis, pp. 225–236 (2014)
3. Frajták, K., Bureš, M., Jelínek, I.: Reducing user input validation code in web applications using Pex extension. In: ACM 15th International Conference on Computer Systems and Technologies, pp. 302–308 (2014)
4. Li, X., Xue, Y.: A survey on server-side approaches to securing web applications. ACM Comput. Surv. (CSUR) **46**(4), 54:1–54:29 (2014)
5. Cho, S., Choi, J., Kim, G., Park, M., Cho, S., Han, S.: Runtime input validation for Java web applications using static bytecode instrumentation. In: ACM International Conference on Research in Adaptive and Convergent Systems, pp. 148–152 (2016)
6. Medeiros, I., Neves, N.F., Correia, M.: Automatic detection and correction of web application vulnerabilities using data mining to predict false positives. In: ACM 23rd International Conference on World Wide Web, pp. 63–73 (2014)
7. Shar, L.K., Tan, H.B.K.: Predicting common web application vulnerabilities from input validation and sanitization code patterns. In: 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 310–313 (2012)
8. Shar, L.K., Tan, H.B.K.: Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities. In: 34th International Conference on Software Engineering, pp. 1293–1296 (2012)
9. Solomon, O.U., William, J.B., Lu, F.: Applied machine learning predictive analytics to SQL injection attack detection and prevention. In: IFIP/IEEE IM 2017 Workshop: 3rd International Workshop on Security for Emerging Distributed Network Technologies, pp. 1087–1090 (2017)
10. Fang, Y., Peng, J., Liu, L., Huang, C.: WOVSQLI: detection of SQL injection behaviors using word vector and LSTM. In: Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, pp. 170–174 (2018)
11. Haldar, V., Chandra, D., Franz, M.: Dynamic taint propagation for Java. In: Annual Computer Security Applications Conference, pp. 09–15 (2005)
12. Halfond, W.G.J., Orso, A.: AMNESIA analysis and monitoring for neutralizing SQL-injection attacks. In: Proceedings of IEEE and ACM International Conference on Automatic Software Engineering, Long Beach, CA, USA, pp. 54–59 (2005)
13. Komiya, R., Paik, I., Hisada, M.: Classification of malicious web code by machine learning. In: 3rd International Conference on Awareness Science and Technology (iCAST), pp. 406–411 (2011)
14. Jingling, Z., Junxin, Q., Liang, Z., Baojiang, C.: Dynamic taint tracking of Web application based on static code analysis. In: 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 96–101 (2016)
15. Kumar, S., Mahajan, R., Kumar, N., Khatri, S.K.: A study on web application security and detecting security vulnerabilities. In: 6th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), pp. 451–455 (2017)

16. Jane, P.Y., Chaudhari, M.S.: SQLIA: detection and prevention techniques: a survey. IOSR J. Comput. Eng. (IOSR-JCE) **2**, 56–60 (2009). Second International Conference on Emerging Trends in Engineering (SICETE)
17. Peng, C.J., Lee, K.L., Ingersoll, G.M.: An introduction to logistic regression analysis and reporting. J. Educ. Res. **96**, 3–14 (2002)