Securing Web Application from SQL Injection Attacks

J. Krishnaveni[1], R. K. Kavitha[2], P. Sujitha[3], T. Jenitha[4]

[1,2,3,4] *Department of Computer Science & Engineering, Mepco Schlenk Engineering College, Sivakasi, Tamil Nadu*

***Abstract*— Software systems, like web applications, are often used to provide reliable online services such as banking, shopping, social networking, etc., to users. The increasing use of such systems has led to a high need for assuring confidentiality, integrity, and availability of user data. SQL Injection Attacks (SQLIAs) is one of the major security threats to web applications. It allows attackers to get unauthorized access to the back-end database consisting of confidential user information. This is an attacking method that aims the data stored in a database through the firewall that shield it. The poor input validation in code and website administration leads SQL injection to attack the web resource. This proposed system exhibit a dynamic method to detect and prevent SQL Injection from malicious web users who wants to access any resource in web related application. To detect malicious attack and prevent malicious users from accessing web resources, this system uses an effective SQL Query processing based on XML Schema validation.**

***Index Terms*— SQL Injection, Web Security, Web application, XML Schema, Dynamic detection.**

I. INTRODUCTION

Database systems are commonly used to provide back end functionality to many types of web applications. In support of web applications, user-supplied data is often used to dynamically build SQL statements that interact directly with a database. A SQL injection attack is an attack that is aimed at subverting the original intent of the application by submitting attacker-supplied SQL statements directly to the back end database. Depending on the web application, and how it processes the attacker-supplied data prior to building a SQL statement, a successful SQL injection attack can have far-reaching implications.

SQL injection (SQLI) is considered one of the top 10 web application vulnerabilities of 2007 and 2010 by the Open Web Application Security Project. In 2013, SQLI was rated the number one attack on the OWASP top ten. In a 2012 study, security company Imperva observed that the average web application received 4 attack campaigns per month, and retailers received twice as many attacks as other industries. SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Due to the nature of programmatic interfaces available, J2EE and ASP.NET applications are less likely to have easily exploited SQL injections. There are 3 ways to perform SQL Injection attacks. SQL Injection attacks can be performed via one of following Database, Network & Webpage. In a database, attacker will do the injection via Query which directly accesses the database for information retrieval. In Network field, attack will be performed by Injecting IP address. For Webpage, attacker chooses a script file for injecting attack to a webpage.

Different kinds of SQLIAs known to date are Tautology, Union, Piggy-Backed Query, Stored Procedures and Alternate Encodings. Attack Intention for tautology type is bypassing authentication, identifying injectable parameters, extracting data. The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The consequences of this attack depend on how the results of the query are used within the application. In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>. Picky-bagged queries, in this attack type, an attacker tries to inject additional queries into the original query. We distinguish this type from others because, in this case, attackers are not trying to modify the original intended query; instead, they are trying to include new and distinct queries that “piggy-back” on the original query. As a result, the database receives multiple SQL queries. The first is the intended query which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first. This type of attack can be extremely harmful.

II. RELATED WORK

In general, there is extensive literature on describing the vulnerabilities in web application. This section reviews about the some related work in order to explore the strengths and drawbacks of existing methods.

Many works have been developed and different tools are available to detect and prevent SQL Injection in web related applications. V.B. Livsh uses static analysis techniques to detect vulnerabilities in software. The basic approach is to use information flow techniques to detect when tainted input has been used to construct an SQL query. The primary limitation of this approach is that it can detect only known patterns of

SQLI attack.

W. G. J. Halfond and A. Orso have proposed an approach that uses static analysis combined with automated reasoning to verify that the SQL queries generated in the application layer cannot contain a tautology. The primary drawback of this technique is that its scope is limited to detecting and preventing tautologies and cannot detect other types of attacks.

In the past, well known Machine Learning technique like learning based anomaly have been proposed to detect and mitigate injection attacks. These techniques monitor the input request and observe the values of an input attributes. They check whether a given input attribute is valid by comparing it against the legitimate model of an input attribute. Based on the result of model, they detect attacks. However, these techniques are only applicable to injection attacks. They do not capture the sequencing of user behavior which leads to more sophisticated attacks like authentication bypass.

Buehrer [11] proposed a technique to avoid SQL Injection attacks by comparing, in runtime, the parse tree of the SQL statement before inclusion of user input with that resulting after inclusion of input. The problem is that this technique depends on changes in the source code and, consequently, on its adoption by the programmers.

Web vulnerability scanners are well-known penetration testing tools that allow checking applications against security issues automatically. Vieir [12] used four commercial scanners to identify security flaws in 300 publicly available web services. The differences in the vulnerabilities detected by each scanner, the low coverage (less than 20% for two of the scanners), and the high number of false positives (35% and 40% in two cases) observed, highlight the limitations of these tools.

Nuno Antunes [13] proposed a new automatic approach for the detection of SQL and XPath injection vulnerabilities in web services code. This approach is based in two main steps. First we generate and run a workload to exercise the web service and learn the profile of the SQL/XPath commands issued. Afterwards it applies a set of command injection attacks and observes the SQL/XPath commands being executed. This allows us to detect existing vulnerabilities by matching incoming commands during attacks with the valid set of commands previously learned.

III. PROPOSED ALGORITHM

Proposed system is used to detect and prevent SQL Injection Attack with runtime monitoring. It entails a new approach for detecting SQL injection vulnerabilities in web application.

Attacker will inject a malicious input via login form as shown in the Figure 3.3.1. After that in the developers side Query will be formed for database access as:

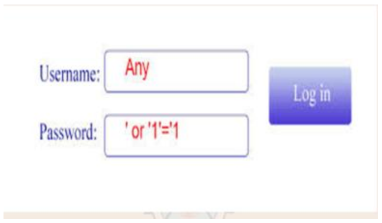

Figure 1: Login Form – SQL Injection Attack

Select * from user where username=‘Any’ and password=‘ ’ or ‘1’=‘1’;

In between web application and database server proposed system created a monitoring system which will evaluate a query formed by developer’s side before accessing the database. To evaluate the proposed approach, a web based e-banking application was developed.

System will create a XML file for each user. Before give the permission to access database, XML file validated by predefined XML Schema file (XSD file). If any vulnerability is detected during validation process, then system will block the user, if not the case then it will provide permission for further process. It is only applicable for known attacks because for known attack only developer created a pattern in XML Schema. But in case of unknown attacks injection will not detected by XML Schema. So system will provide a second level of authentication for unknown attack. During unknown attack, an automated file will be created which contains details about new attack. That file will be sent to the administrator mail id for analyzing.

IV. SYSTEM DESIGN

Figure 2 illustrates the overall system architecture of the proposed system. The web user gives their information as input on web page. The main aim is to detect the SQL injection vulnerability in proposed system. If any vulnerability detected in given input, then that should be prevented. The given inputs are checked in server side before getting database access.

In Query parser, the given input is parsed and retrieves the user information from the query string. The user information is generated as XML file by using XML File Generator. For each value in array of input is created as user defined tags.

XML file will be created for each user who trying to access the web page. After the session, XML file which is suspected for SQL Injection vulnerability alone maintained other than that XML file created for wrong and valid user are deleted for efficient space usage.

At first we have predefined XML Schema file. In XML Schema file, all possible combination of SQL injection vulnerability patterns is restricted. After creating XML file, that validated by xml schema file. XML file analyser analyse the xml file by using Schema file. It checks any SQL Injection vulnerability occurred on web applications. If any injection possibility is detected, later that will informed to the administrator.

Query Processor module is used to detect the new vulnerability of SQL injection. If any vulnerability detected while validation process, then block the access of the user. Any unknown attack is detected, that will be stored in a separated file called Error log.

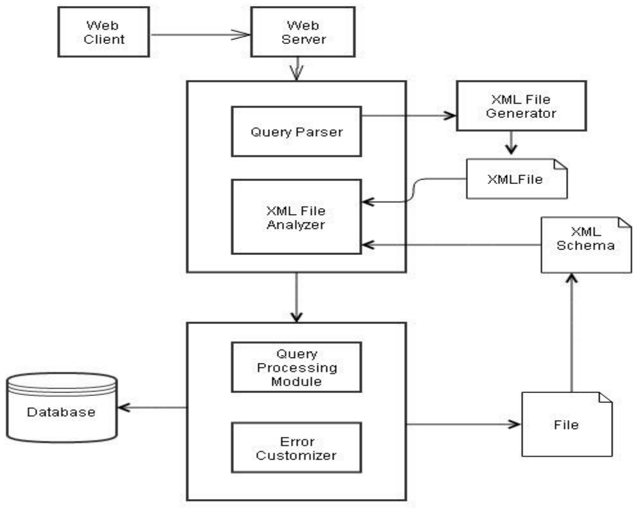

Figure 2: Overall System Architecture

Error customizer customizes the error type that occurred as unknown attack. It saves the error information about newly occurred SQL Injection attack in error log. If any new SQL Injection attack is detected, then that error log file is mailed to the administrator. Finally, administrator can update the XML Schema file according to their error information.

If any vulnerability is detected during validation process, then system will block the user, if not the case then it will provide permission for further process. It is only applicable for known attacks because for known attack only developer created a pattern in XML Schema. But in case of unknown attacks injection will not detected by XML Schema. So system will provide a second level of authentication for unknown attack. During unknown attack, an automated file will be created which contains details about new attack.

4.1 XML File Creation

It is used to create a xml file by the user given inputs. The web user gives their information on web page. Array is created to storing the list of user information. For each information stored in that array, a tag is created. After creating the tags, xml file is generated with root tag. In xml file, the user information is constructed as user defined tags.

4.2 XML File Validation

It validates the xml file for SQL injection attack, xml file validated against the xml schema (.xsd file) which is already developed by an administrator. XML schema contains a restriction pattern for tautology, union and piggy packed queries. XML Schema only developed for known attack. If XML File contains any values Which is restricted by XML Schema then it will block the user access. If the tags do not contain any vulnerability then it will redirect to the validation for authentication purpose. If the user is a valid person then system will displays their personal details, else it will redirect to home page.

If the XML file is against xml schema (i.e.) contains SQL injection vulnerability then system will maintain that xml file for further use, if not the case then it will unlink the file (i.e.) delete the file from the collections.

4.3 Unknown Attack Detection

XML Validation against XML schema is only applicable for known attacks which have a restricted pattern in xml schema file. For detecting unknown attack system will create a second level of authentication. Because unknown attack will not be detected by XML schema so after doing XML validation and authentication process for valid user before going to the database access system will create an another firewall for unknown attacks. Retrieved values are validated by the database values. If any vulnerability is detected then system will create a file which has the details about unknown attack. Date and time of occurrence set to be a file name.

4.4 Schema Updation

In case of unknown attack system will create a file which contains attack information for further process. For avoid the same problem in future system should update this attack in to XML Schema if it will update a restricted pattern for unknown attack then next time of occurrence will be restricted at first level itself (XML Schema validation). For that system will send a notification via mail to administrator. Administrator will update the XML schema with use of file created for that attack.

V. DISCUSSIONS

The web application vulnerability has been identified in a particular website. We are creating a website to detect the vulnerabilities and to prevent the website from SQLI vulnerabilities. In this system, we create a XML file for each user login into the website. Then it checks the vulnerabilities in XML file by using XML Schema file. After it detects the vulnerabilities and block the user if any vulnerability detected. If any new vulnerability detected by our system, then that will be updated in XML Schema for future use.

VI. CONCLUSION

In proposed system, a technology for securing web application from SQL Injection attacks. Proposed technology only applicable for Tautology, Union and Piggy backed Query type SQL Injection attack. The User information’s are taken from form elements and using that information XML file was created. Using a predefined XML Schema XML file was validated if the file contains any known attack system will block the user, else it give permission to access the database. Before fetching values from the database there is a second level validation for detecting unknown attacks. If the file contains any vulnerability then it blocks the user and creates the error file. The error file was sent to the administrator through mail. Administrator check the error file if it contains any vulnerability then the admin updates the XML Schema for future use.lthough a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions. Authors are strongly encouraged not to call out multiple figures or tables in the conclusion—these should be referenced in the body of the paper.

VII. Future Enhancement

In future, system can be extended to all types of SQL Injection attacks. Actually SQL Injection has four types of attacks but system implemented three of them. In future system will more concentrate on stored procedure. In the proposed system XML schema was updated only by administrator so manual update will cause time consuming process. Future enhancement will provide a solution for that. So in future it can be done as update the XML Schema dynamically when the unknown attacks identified.

References

[1]. Lwin Khin Shar and Hee Beng Kuan Tan, "DEFEATING SQL INJECTION", vol. 46, Issue : 3 , IEEE Computer Society, ISSN : 0018-9162, page numbers: 69-77 28 March 2013

[2]. Y. Shin, L. Williams, and T. Xie, *SQLUnitGen: Test Case Generation for SQL Injection Detection*, tech. report TR 2006-21, Computer Science Dept., North Carolina State Univ., 2006.

[3]. H. Shahriar and M. Zulkernine, "MUSIC: Mutation-Based SQL Injection Vulnerability Checking," *Proc. 8th Int'l Conf. Quality Software* (QSIC 08), IEEE CS, 2008, pp. 77-86. J. Fonseca, M. Vieira, and H. Madeira, "Vulnerability & Attack Injection for Web Applications," *Proc. 39th Ann. IEEE/IFIP Int'l Conf. Dependable Systems and Networks* (DSN 09), IEEE, 2009, pp. 93-102.

[4]. J. Fonseca, M. Vieira, and H. Madeira, "Vulnerability & Attack Injection for Web Applications," *Proc. 39th Ann. IEEE/IFIP Int'l Conf. Dependable Systems and Networks* (DSN 09), IEEE, 2009, pp. 93-102.

[5]. S.W. Boyd and A.D. Keromytis, "SQLrand: Preventing SQL Injection Attacks," *Proc. 2nd Conf. Applied Cryptography and Network Security* (ACNS 04), LNCS 3089, Springer, 2004, pp. 292-302.

[6]. Ramya Dharam and Sajjan G. Shiva, "Runtime Monitoring Technique to handle Tautology based SQL Injection Attacks", ISSN: 2305-0012, page number : 189-203, International Journal of Cyber-Security and Digital Forensics (IJCSDF).

[7]. Stephen W. Boyd, Gaurav S. Kc, Michael E. Locasto, Angelos D. Keromytis, and Vassilis Prevelakis, "On The General Applicability of Instruction-Set Randomization"

[8]. V.B. Livshits and M. S. Lam. Finding Security Errors in Java Programs with Static Analysis. In Proceedings of the 14th Usenix Security Symposium, pages 271–286, Aug.2005.

[9]. W. G. J. Halfond and A. Orso, "Combining Static Analysis and Runtime Monitoring to Counter SQL Injection Attacks," 3rd Intl. Workshop on Dynamic Analysis, 2005, pp.1- 7

[10]. D. Q. Naiman, Statistical anomaly detection via httpd data analysis, Computational Statistics & Data Analysis, Vol.45, Issue.1, pp.51-67, 2004.

[11]. Buehrer, G., Weide, B., Sivilotti, P., "Using Parse Tree Validation to Prevent SQL Injection Attacks", International Workshop on Software Engineering and Middleware, 2005.

[12]. Vieira, M., Antunes, N., Madeira, H., "Using Web Security Scanners to Detect Vulnerabilities in Web Services", Intl.Conf.on Dependable Systems and Networks, Lisbon, 2009.

[13]. Nuno Antunes, Nuno Laranjeiro, Marco Vieira, Henrique Madeira "Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services", IEEE International Conference on Services Computing, 2009.