www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 1, Jan 2014 Page No. 3695-3699

An Approach to Detect and Prevent Tautology Type SQL Injection in Web Service Based on XSchema validation

***R. Joseph Manoj*[1] *Dr.A.Chandrasekhar*[2] *M.D.Anto Praveena*[3]**

[1]Research Scholar, Manonmanium Sundaranar University, Tirunelveli, India &
Associate Professor, St.Joseph's College of Engineering, Chennai, India, rjmanoj79@gmail.com.
[2]Professor, St.Joseph's College of Engineering, Chennai, India, drchandrucse@gmail.com.
[3]Assistant Professor, Sathyabama University, Chennai, India, antopraveena@gmail.com.

Abstract: ***SQL injection is the most common attack for web applications and widely used attacking method by hackers all over the world. 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 tautology type 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. This proposed system can be used in web application logging and security. This work also concludes effectiveness and performance of the system using the resultant data of proposed system.***

Keywords: SQL injection, web services security, web service authentication.

1. Introduction

Web applications are nowadays a strategic mean for data exchange and systems integration as they afford a simple interface between a web service provider and a web service consumer [1]. However, web services are so widely exposed that any existing security vulnerability will most probably be revealed and oppressed by hackers.

The World Wide Web (WWW) has experienced noteworthy growth in recent years. Business organizations, individuals and government organizations have found that web applications can offer effective, efficient and trustworthy solutions to the challenges of communicating and conducting commerce in the 21th century [2]. Various corporate bodies like eBay, Google, Yahoo, Amazon business model entirely focuses on the web. As more and more enterprise applications dealing with responsive financial and confidential data turn online, the security of such Web applications have come under secure inspection. Compromise of these applications represents a serious threat to the organizations that have deployed these web applications as well as to the users that trust these systems to store confidential data.

1.1 SQL Injection and techniques

SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. Any process that constructs SQL statements should be reviewed for injection vulnerabilities because SQL Server will execute all syntactically valid queries that it receives. It has become a common issue with database-driven web sites. This flaw is easily detected, and easily exploited, and as such, any web related application with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes. The fear of SQL injection attacks has become increasingly frequent and serious. SQL-Injection Attacks are a class of attacks that many of these systems are highly vulnerable to, and there is no known fool-proof defend against such attacks [3].
Example: Consider the following query in the ASP page is of the form;
*SELECT * FROM EMPLOYEE WHERE NAME='$login' AND PASS='$pass'*
If the login and password as provided by the user are used, the query to be submitted to the database takes the form; *SELECT * FROM EMPLOYEE WHERE NAME='abc' AND PASS='xyz'*

A web site that uses this asp would be vulnerable to SQLIAs. If the user were to enter [' OR 1=1 --] and [ ] instead of [abc] and [xyz], the query would take the form; SELECT PROFILE FROM EMPLOYEE WHERE NAME=' ' OR 1=1 --' AND PASS=' '. The characters "--" mark the beginning of a comment in SQL, and everything after that is ignored The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology the query evaluates to true for each row in the table and returns all of them. Thus an attacker can bypass all authentication modules in place and gain unrestricted access.

There are many techniques that are used to perform SQL Injection attacks in web related applications. Very few of the techniques are given below [4]:

a. Tautologies

Tautology-based attack is to inject code in one or more conditional statements so that they always assess to true. The most common usages of this technique are to bypass authentication pages and extract data. If the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned.

b. Union Query

In union-query attacks, Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query> because the attackers completely control the second/injected query they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

c. Stored Procedures

A stored procedure is an operation set that is stored. Typically, stored procedures are written in SQL. Since stored procedures are stored on the server side, they are available to all clients. Once the stored procedure is modified, all clients automatically get the new version.

d. Alternate Encodings

Alternate encodings do not provide any unique way to attack an application they are simply an enabling technique that allows attackers to evade detection and prevention techniques and exploit vulnerabilities that might not otherwise be exploitable. To evade this defence, attackers have employed alternate methods of encoding their attack strings like hexadecimal, ASCII, and Unicode character encoding. Therefore, attackers have been very successful in using alternate encodings to conceal their attack strings.

e. Deny Database service

This attack used in the websites to issue a denial of service by shutting down the SQL Server [5]. A powerful command recognized by SQL Server is SHUTDOWN WITH NOWAIT. This causes the server to shutdown, immediately stopping the Windows service.

The rest of the paper is organized as follows: The related study of proposed work is discussed in section 2. In Section 4 and 5, proposed system and implementation results are discussed. Finally the paper concludes the paper in Section 6.

2. Related Study

Many works have been developed and different tools are available to detect and prevent SQL Injection in web related applications. Livshits and Lam [6] use 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.

Wassermann and Su 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 [7]. The primary drawback of this technique is that its scope is limited to detecting and preventing tautologies and cannot detect other types of attacks.

AMNESIA [8] is a model-based technique that combines static analysis and runtime monitoring. In its static phase, AMNESIA uses static analysis to build models of the different types of queries an application can legally generate at each point of access to the database. In its dynamic phase, AMNESIA intercepts all queries before they are sent to the database and checks each query against the statically built models. Queries that violate the model are identified as SQLIAs and prevented from executing on the database. The primary limitation of this technique is that its success is dependent on the accuracy of its static analysis for building query models

Valeur and colleagues [9] propose the use of an Intrusion Detection System (IDS) to detect SQLIA. It is based on a machine learning technique that is trained using a set of typical application queries. The technique builds models of the typical queries and then monitors the application at runtime to identify queries that do not match the model in that it builds expected query models and then checks dynamically-generated queries for compliance with the model. Their technique, however, like most techniques based on learning, can generate large number of false positive in the absence of an optimal training set.

Cheng [10] presented user behavior surveillance system, a novel Embedded Markov Model to detect various Web application attacks. This model not only considers injection attack but also captures the sequencing of user behavior. Their model can detect unreasonable transition of user behavior and mitigate authentication bypass attack.

In the past, well known Machine Learning technique like learning based anomaly [11] 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, G [12] is 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. Vieira, M et.al (2009) [13] 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 et al (2009) [14] 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.

Indrani Balasundaram [15] paper proposes a novel specification-based methodology for the prevention of SQL injection Attacks. The two most important advantages are first, it prevents all forms of SQL injection attacks; second, current technique does not allow the user to access database directly in database server.

Shanmughaneethi, V [16] proposed a new method for the detection of XPath injection vulnerabilities in XML database. In this method a XPATH query is converted into XML document. This XML document will be checked with already defined XML schema for validness. If the XML document passes, XPATH query has no injection otherwise will be considered as XPATH injection and the corresponding user not be allowed to continue the process.

3. Proposed System

This Technique is used to detect and prevent SQL Injection Attack with runtime monitoring. The proposed system entails a new approach for detecting SQL injection vulnerabilities in web application shown in Figure.1.This approach incorporates SQL expression scanner, XML file maker and XSchema validator. The below architecture describes the SQL injection detection technique proposed in this system. .The working of all these modules given as follows:

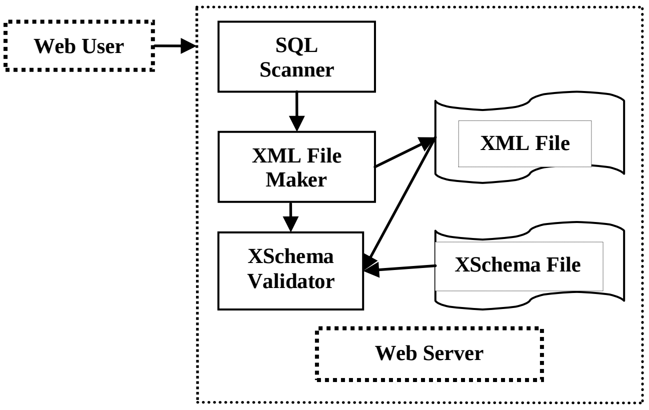

Figure 1: Proposed System Architecture

3.1. SQL Scanner

Query scanner involves intercepting the SQL query that is generated at run time. In order to detect SQL injection vulnerabilities, this dynamically generated expression has to be intercepted. Also intercepting the run time query can be used to detect any type of injection, since the sink points for causing the injections are these queries that are generated at run time. Using SQL scanner, these run time generated queries can be intercepted before they are executed in the database server.

3.2 XML File Maker

In this module the intercepted SQL Query is analyzed and XML file is created by obtaining the input parameters to detect possible injections. After intercepting the query, the analyzer obtains the inputs from the SQL query and stores them in a XML document. This document is then further used for validation in order to detect vulnerabilities. Consider the SQL query for login mode: *select * from user_Info where username='John' OR 1=1 - -'password=''*

The below Figure 2 shows a sample XML file that would be generated after the above SQL query is intercepted. This XML file consists of only the input parameters that were given as user inputs from the client application. Further this can be used for validation in order to find if any injection is present.

```
<?xml version="1.0" encoding= "utf-16"?>
<sqlxper>
<inp1>John</inp1>
<inp2>'1' ='1</inp2>
</sqlxper>
</xml>
```

Figure 2: Sample XML file for the SQL Query

3.3. XSchema Validator

This validation process is to identify the injected parameters with the help XSchema and the generated XML file in previous phase. The schema is a generalized metadata which define structure and type of user input. So in this approach, a well defined XSchema is defined for detecting possible injection characters in the input values provided by the user.

The XML file which is generated from the previous module that consists of the user input is now validated with a well defined XML schema. If the validation passes then injection is not present in the input parameters, in case of failure the injection is logged in a log file. The log file clearly indicates the attack input mismatch with the schema thereby avoiding the injection to take place.

```
<xs:element name="inp1">
    <xs:simpletype>
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-zA-Z0-9]>
        <xs:restriction>
    <xs:simpletype>
<xs:element>
<xs:element name="inp2">
    <xs:simpletype>
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-zA-Z0-9]{8}>
        <xs:restriction>
    <xs:simpletype>
<xs:element>
```

Figure 3: Sample XML Schema for the XML Validation

If the validation process is passed, then the operation is allowed to access the desired resource and results are obtained. The schema is vital in detecting injections in the input. Input can be of any type and hence the schema restricts values for each data type thereby providing an effective validation process.

4. Performance Evaluation

To evaluate the proposed approach, a web based e-library application was developed. Then performance of the system based on the response time is analyzed for tautology based SQL Injection. The response time in real web environment is collected and tabulated is shown in Table1 [15]. The table concludes that proposed system execution time is better than existing system. But it detected all tautology type SQL Injection and prevents malicious attacks given by different web users.

TABLE 1: Execution Time Comparison of Proposed System

Number of Test	Execution Time in seconds	
	Existing system	Proposed System
1	27.33	25.44
2	23.66	22
3	17.33	16.55
4	20.16	19.75
5	27.83	26.33

Table 1 illustrates that execution time of proposed system is better than existing system. Since this proposed system is a XML based approach which can be used widely in web related security and logging, etc. The following graph in Fig 4 shows the comparison of the execution time, between the proposed system and without it. As shown in the graph, the system does not bring an enormous difference in the execution time. In the following graph *X* – Axis represents Number of Test and Y – Axis represents Response Time. The pictorial representation of performance evaluation is given in Figure 4 below

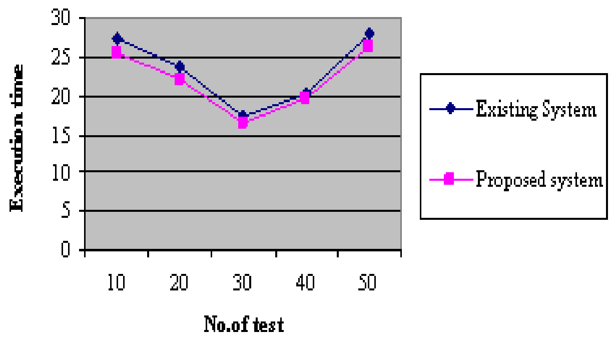

Figure 4: Execution Time comparison for proposed technique with existing technique

5. Conclusion and Future Enhancement

This paper has proposed a new approach for detecting tautology type SQL injection. This kind of approach is a very effective method for detecting malicious attacks in web applications. Comparing with previous approaches, there is a significant improvement in execution time and also protect the system from the tautology based SQL Injection in web related logging system. In future, this paper will analyze other types of SQL injection like Cross-site scripting and stored procedure. Also this system can be extended as an efficient authentication system which can authenticate web users and prevent malicious users.

References

[1] Curbera, F. et al., “Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI”, Internet Computing, IEEE, vol. 6, pp. 86-93,2002.

[2] Structured query language (SQL) related tools-Ke Wei, 2012.

[3] Li Sha Dong Xiaori RaoHong, “An Adaptive Method Preventing Database from SQL Injection Attacks”, IEEE International Conference on Services Computing 2010

[4] Rahul Shrivastava, Joy Bhattacharyji, Roopali Soni, “ Recognition and Deterrence of SQL injection attacks in database using web service”, International Journal of Software and Web Sciences (IJSWS), ISSN (Online): 2279- 0071 ,PP: 8-16

[5] Nils Gruschka, Norbert Luttenberger, “Protecting Web Services from DoS Attacks by SOAP Message Validation”, 2012

[6] 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.

[7] 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

[8] William G. J. Hal fond and Alessandro Orso, “AMNESIA: Analysis and Monitoring for Neutralizing SQL Injection attacks”.

[9] F.Valeur and D. Mutz and G. Vigna “A Learning-Based Approach to the Detection of SQL Attacks,” In Proceedings of the Conference on Detection of Intrusions and Malware Vulnerability Assessment (DIMVA), July 2005.

[10] Cheng, Y., Laih, C., Lai, G., Chen, C., and Chen, T. 2008. Defending On-Line Web Application Security with User-Behavior Surveillance. Intl.Proceedings of the 2008 Third international Conference on Availability, Reliability and Security (March 04 - 07, 2008). ARES. IEEE Computer Society, Washington, DC, 410-415.

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

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

[13] 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.

[14] 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.

[15] Indrani Balasundaram, Dr. E. Ramaraj “An Approach to Detect and Prevent SQL Injection Attacks in Database Using Web Service”, IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, January 2011.

[16] Shanmughaneethi, V., R. Ravichandran, and S. Swamynathan. "PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications." International Journal on Web Service Computing 2.3, 2011.

Author Profile

Joseph Manoj.R, received his MCA degree from Annai Velankanni College affiliated to Manonmanium Sundaranar University, Tirunelveli, India. M.E(CSE) degree from Sathyabama University, Chennai, India and Pursuing Ph.D in Computer science & Engineering in Manonmanium Sundaranar University, Tirunelveli, India. He is currently working as a Associate Professor in the

Department of MCA in St.Joseph's College of Engineering, Chennai. His research area is web services security. He is a member of ACM.

Chandra Sekar A, received his B.E(CSE) degree from Angala Amman College of Engineering and Technology affiliated to Bharathidasan University, M.E(CSE) degree from A.K. College of Engineering affiliated to Madurai Kamaraj University, and Ph.D in Information and Communication Faculty (Computer science & Engineering) from Anna University, Chennai, India. He is currently working as a professor in the Department of Computer Science & Engineering in St.Joseph's College of Engineering, Chennai. His area of interest includes Network Security, web services and Analysis of Algorithms.