Preventing SQL Injection Attacks

Article *in* International Journal of Computer Applications · August 2012

DOI: 10.5120/8264-1809

CITATIONS
11

READS
180

3 authors, including:

Asha Nathan
VIT University
23 PUBLICATIONS **121** CITATIONS
SEE PROFILE

Varun Kumar M
VIT University
51 PUBLICATIONS **57** CITATIONS
SEE PROFILE

Preventing SQL Injection Attacks

Asha. N
Assistant Professor
School of Information Technology and Engineering
Vellore Institute of Technology, Vellore, India

M. Varun Kumar
Assistant Professor
School of Information Technology and Engineering
Vellore Institute of Technology, Vellore, India

Vaidhyanathan.G
M.S. Software Engineering
School of Information Technology and Engineering
Vellore Institute of Technology, Vellore, India

ABSTRACT
With the recent rapid increase in web based applications that employ back-end database services, results show that SQL Injection and Remote File Inclusion are the two frequently used exploits rather than using other complicated techniques. With the rise in use of web applications, SQL injection based attacks are gradually increasing and is now one of the most common attacks in the internet. It allows an attacker to gain control over the database of an application, thereby able to read and alter confidential data. This paper illustrates few different forms of SQL injection and based on observation, it is seen that SQL Injection is interpreted differently on different databases. Finally, an effective solution is proposed for the prevention of these kinds of injection attacks, in such a way that it is independent of the underlying platform and database. Two levels of user authentication has been proposed in this method, SQL based authentication and an XML based authentication, and has been found to be very effective in preventing such attacks.

General Terms
Web application, Web Security, Authentication, Attacker.

Keywords
Web architecture, SQLIA , HTTP, XML.

1. INTRODUCTION
SQL Injection Attack (SQLIA) is considered one of the top 5 web application vulnerabilities by the Open Web Application Security Project (OWASP) in the year 2010. A Database is an essential component that is necessary in modern web applications. Every web based application that is developed and deployed over the internet, requires the interaction of a database, thereby the application becomes fully database driven. It has been noted that, at an average, applications experience, 71 attempts an hour. Some applications experience aggressive attacks and at a peak, were attacked 800–1300 times per hour.

An SQL injection attack involves the insertion or "injection" of a SQL query by an attacker via the input data from the client to the application. This injection in the SQL query involves inserting malicious input statements by an attacker. The execution of these malicious input statements by the web server at the database end results in unexpected behaviour thus compromising the security of the database. The database just executes the input data provided by a client/attacker as it is. It does not have the ability to differentiate between a valid input string and/or a malicious/injected input string.

A successful SQL injection exploit can

- read sensitive data from the database
- alter database data (Insert/Update/Delete)

1.1 Modern Web Architecture
The diagram below illustrates the general web architecture. Any web based architecture typically follows the Client-Server architecture.

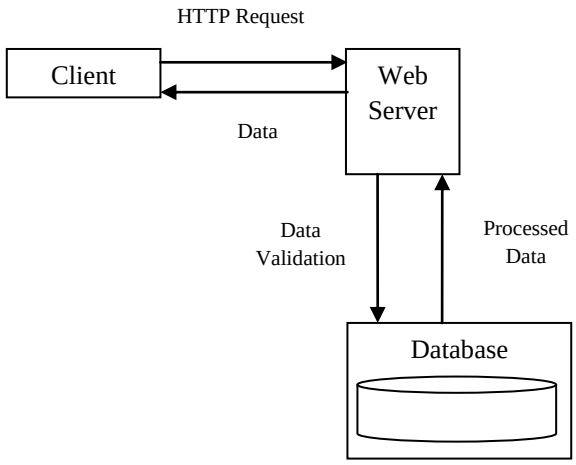

Fig 1.1 Web Architecture

The client sends a HTTP request to the Web Server. This request will have the user input data. This input data will be sent to the database layer for processing by the web server. At the database end, the SQL queries will be processed and the results will be sent to the web server.

Hence, the entire web application is database-driven. The database server usually contains many databases, and in turn, each database contains many tables. The database is under huge threat to the attackers.

1.2 Intent
The attacker's objective for using the injection technique lies in gaining control over the application database, thereby able to access and alter confidential data for which he is not authorized to. By gaining access to unauthorized data, information pertaining to other users is available to the attacker, thereby the system fails to ensure data confidentiality and integrity.

In a web based application environment, the number of users accessing the system is huge. Most of the web based applications, social websites, banking websites, online shopping websites require a user to sign up for the system. This usually involves creating user credentials such as a user identity and a password. A user is identified by the system based on his identity. This process of validating an individual based on a username and password, is referred as authentication. The system identifies the user, and provides

individual access to the system objects based on his/her identity. This process is referred as authorization. This user credential and other information is stored in the database, and accessing this database is the main goal of the attacker.

From the perspective of an attacker, the main intentions for applying this technique is to be able to

- Bypass authentication procedure
- Extract the existing data from the system (Confidentiality lost)
- Alter the existing data (Integrity lost)

Bypassing an authentication is a serious threat, as it allows an attacker to forge another authorized user identity, perform certain actions on behalf of the other user, and importantly access/modify confidential information that belongs to the user.

This paper mainly focuses on how the attacker uses various injection techniques to bypass the authentication procedure in a system and presents an algorithm for prevention of such attacks.

2. LITERATURE SURVEY

A survey of all recent research works done in the field of SQL injection attacks has been listed out. A lot of research has been done in detecting and preventing injection attacks and few approaches are discussed below.

[1] The system proposed by Mehdi Kiani, Andrew Clark and George Mohay uses an anomaly based approach which utilizes the character distribution of certain sections of HTTP requests to detect previously unseen SQL injection attacks. The advantage of the system proposed by Mehdi Kiani et.al is that it does not require any user interaction, or no modification of, or access to the backend database or the source code of the web application. The problem faced is the high rate of false alerts which had to be taken care while implementing the system in real time environment. This is because of less information available on attacks to the administrator, thus making it difficult to differentiate between false alerts and the real attacks. [2] V.Shanmughaneethi, C.Emilin Shyni and Dr.S.Swamynathan uses a methodology which make use of an independent web service which is intended to generalize the syntactic structure of the SQL query and validate user inputs. The SQL query inputs submitted by the user are parsed through an independent service and the correctness of the syntactic structure of the query is checked. The main advantage of this paper is that the error message generated doesn't contain any Meta data information about the database which could help the attacker. Since the web service is not integrated with the web application, any modification that should be done to the system should be done in such a way that it should be supported by the web service.[3] R. Ezumalai, G. Aghila, proposed a combinatorial approach for shielding web applications against SQL injection attacks. This combinatorial approach incorporates signature based method, used to address security problems related to input validation and auditing based method which analyze the transactions to find out the malicious access. This approach requires no modification of the runtime system, and imposes a low execution overhead. It can be inferred from this approach that the public interface exposed by an application becomes the only source of attack.[4] Yuji Kosuga, Kenji Kono, Miyuki Hanaoka, et.al proposed a technique called Sania for detecting SQL injection vulnerabilities during the development and debugging phases of a web application. It identifies the vulnerable spots by analyzing the SQL queries issued in response to the HTTP requests in which an attacker can insert arbitrary strings. The main feature of Sania is the generation of attacks using the knowledge by this model, thus checking if the SQL injection vulnerabilities lie in the web application. [5] Ke Wei, M. Muthuprasanna, Suraj Kothari proposed a technique to defend attacks against the stored procedures. This technique combines a static application code analysis with a runtime validation to eliminate injection attacks. In the static part, a stored procedure parser is designed, and for any SQL statement that depends on user inputs, and use this parser to instrument the necessary statements in order to compare the original SQL statement structure to that including user inputs. The underlying idea of this technique is that any SQLIA will alter the structure of the original SQL statement and by detecting the difference in the structures, a SQLIA can be identified. [9] Kai-Xiang Zhang, Chia-Jun Lin, et.al proposed a translation and validation(TransSQL) based approach for detecting and preventing SQL Injection attacks. The basic idea of this approach relies on how different databases interpret SQL queries and those SQL queries with injection. After detailed analysis on how different databases interpret SQL queries, Kai-Xiang Zhang, et.al proposed an effective solution TransSQL, using which the SQL requests are executed in two different databases to evaluate the responses generated.

3. OVERVIEW OF THE SYSTEM

SQL injection is a technique that exploits a security vulnerability occurring in the database layer of an application. The key behind this attack is that it alters the structure of the original SQL statement when malicious input statements are added along with the original query.

In this scenario, on bypassing authentication, the injection technique is carried out on login forms where a user has to provide a username and a password, and any other places that has to be provided with a user input. In this paper, we focus the attacker's concentration on a user login form in any web page. A typical login form will contain a username and a password field, and this is where the attacker keeps trying different injection techniques until he compromises the security of the database.

3.1 Consequences of SQLIA

With SQL injections, attackers can take complete remote control of the database, and some of the impacts are:

- Insert a command to obtain access to all account details in the system, including usernames and passwords.
- With the username and password in attacker's hand, he can alter the password; change the privilege of the account.
- Forge an user identity
- Shut down a database.
- Upload files.
- Delete a database and its entire contents.

4. TYPES OF SQLIA TECHNIQUES

4.1 Tautology

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 most common usages are to

bypass the authentication pages and to extract the data. In this type of injection, an attacker exploits an inject able field that is used in query's 'where' conditional. Typically, 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.

A typical user authentication SQL statement at the database end will take the following form.

Eg. Select name from users where name='$name' and pass='$pass';

In the case of a legitimate user, with username as 'user1' and password as 'pass1', the query will take the form,

Select name from users where name='user1' and pass='pass1'; [No Injection]

And when these user credentials are validated by the database, the user is authenticated.

Now as an example of a tautology attack, the attacker submits the malicious code [' OR '1'='1] as input for the username and password field, and the query takes the form,

select name from user where name= ' 'OR '1'='1' and pass= ' 'OR '1'='1 ';

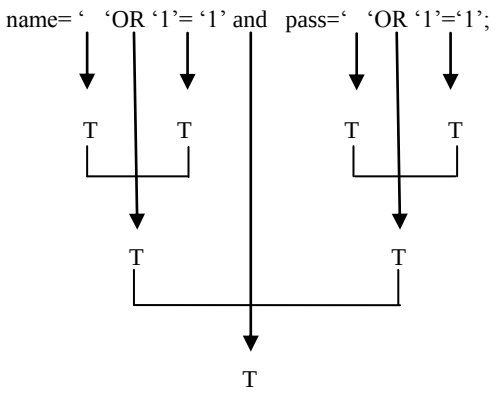

Figure 4.1 Tautology

The code injected in the condition [' OR '1'='1] transforms the entire 'where' clause into a tautology. Since the conditional is a tautology, the query evaluates to a true for each row in the table and returns all of them, and finally the attacker will be authenticated into the system with the identity of the first record returned by the SQL query.

4.2 Logically Incorrect Query

This attack lets an attacker gather important information about the type and structure of the back-end database of a web application. The attack is considered to be an information gathering step for other types of attacks. The vulnerability leveraged by this attack is that the default error page returned by the application servers is often overly descriptive. Such error messages generated can often reveal vulnerable/inject-able parameters to an attacker. When performing this attack, an attacker tries to inject statements that cause a syntax, or logical error into the database.

4.3 Piggy Backed Query

This kind of attack appends additional queries to the original query string. If the attack becomes successful, the database receives and executes a query string that contains multiple distinct queries. The first query is usually the original legitimate query, whereas the subsequent queries are the injected/malicious queries. This type of attack can be harmful because attackers can use it to inject virtually any type of SQL command.

Eg. By using the other injection techniques discussed above, the attacker will have the name(s) of authorized user(s). For subsequent trials and in the case of Piggy Backed queries, he uses the authorized user name as input for the user field, and uses the following malicious code for the password field,

pass = ' OR (SELECT COUNT(*) FROM user)=15 AND "='

The entire SQL statement will take the form,

Select name from users where name='user1' and pass= '= ' OR (SELECT COUNT(*) FROM user)=15 AND ' '=' ' ;

If this query evaluates to true, then the attacker gains an insight that, there are exactly 15 users in the system. If it is evaluated to be false, then the condition is found to be incorrect and tries different possible techniques. Here, if the 'INSERT into' clause is used, and if the condition evaluates to be true, then the attacker can successfully insert data into the database.

5. PROPOSED ALGORITHM

5.1 Existing Technique

In the existing web applications, authentication process takes place as follow. The user enters his assigned user name and password. The database checks if the particular user name, password combination exists in the database, and if it exists, authenticates the user. If we look at the tautology based attack, an attacker might be able to break into the system even without entering a valid user name in the user field. This is the main issue in few of the existing web applications, that there is no proper authentication procedure. This necessitates the need for a strong user authentication procedure.

This algorithm presents an efficient user authentication procedure, in a way that, an input SQL statement will be processed by the database only if the user is found to be a valid user of the system. This totally isolates the database from such injection attacks. A user is validated against two different databases of the same system.

5.2 Proposed Technique

The proposed methodology here is to provide two levels of User authentication at the database level.

1. SQL Authentication
2. XML Authentication

The HTTP request sent by the Client is passed to the Web server. The input user credentials entered in the form are passed to the web server for processing at the database end. Now, the Web server has to pass it to the database. In any form of a SQL based database, Relational Database management system is used.

The problem faced here is that, the same SQL query no matter in which relational database it is executed, it does not have the

ability to differentiate the response or result obtained from the query processed by the database. That is, if a particular SQL request is evaluated to be true in a database, then the same request would evaluate to true on all the other SQL based databases, which happens because all of these databases work based on the relational database management systems.

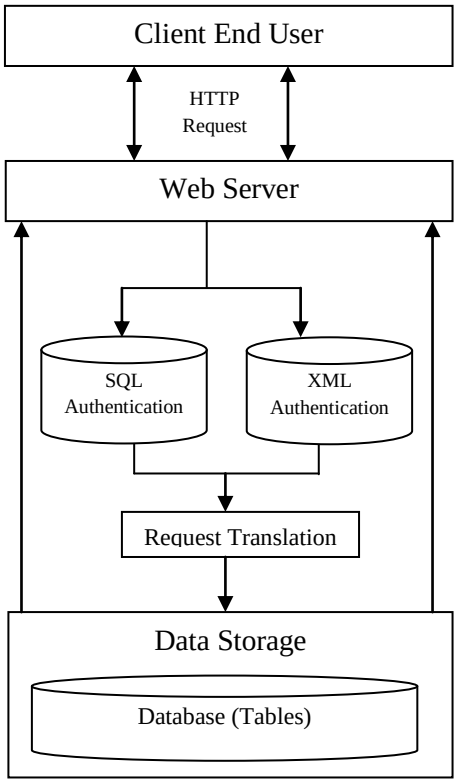

Fig 5.1 Proposed System Architecture

Therefore, if the same malicious/injected SQL request is run on hierarchal based database management system, then the response would be different. In a relational database management system like Microsoft Access, SQL Server, MySQL, data is stored in the form of rows and columns in tables, whereas in a hierarchical database, data is stored in the hierarchical tree structures, with the bottom most nodes that store the value. Hence the way of data processing among relational and hierarchical database management will differ, and this is the core concept of this work.

5.2.1 Using XML

Though XML is a widely used language for transportation of data in the web, there have many instances of using the XML language as a means of just storing the data, thus acting as a database. Also XML, stores data in hierarchical structures of trees, that stores data in terminal nodes, with each of the node constituting a root node. The major advantage of using XML is that, it is widely portable, platform independent and can be integrated very easily into different web technologies. Other existing hierarchical database management systems like Microsoft Active Directory, Apache Directory Studio, Open LDAP for Windows are not as flexible as XML, and require a lot of overhead in initial configuration and, not totally reliable in terms of compatibility.

So, the idea proposed is to replicate or make a copy of the SQL user database in an XML database format. But the retrieval time in a hierarchical database is slow as compared to relational databases if the number of users of the system is high. This is because when data is searched in a XML or hierarchical database, all the nodes in present in the database from the beginning are searched and this consumes a lot of time. So, instead of storing the entire user database in a single database file, a single XML file is created for every user of the system, and the corresponding password alone is stored in the XML file. This reduces the search complexity to a huge extent and also the size of individual file is minimal.

5.3 Preventing SQLIA

So, when a user tries to gain access to a system, initially, SQL authentication is done, which might evaluate to true, even in the case of an input by an attacker. But during the XML authentication, initially the corresponding XML database file is searched in the system, and if present the password is checked and only then is the user validated. So, the attacker can hack into a system only in the case of a authorized username. So in the case of an attacker input/injection, even if the SQL authentication evaluates to true, the XML authentication will fail and hence the request will not be processed by the database, thus preventing the direct access to the database. This is the method implemented in this work.

6. CONCLUSION AND FUTURE WORK

SQL injection is a common technique hackers employ to attack these web based applications. SQL Injection attack has also been specified under the top five web security threats by the Open Web Application Security Project in the year 2010. These attacks reshape the SQL queries, thereby altering the behavior of the program for the benefit of the hacker. In the work carried out, a method is put forward to detect and prevent SQL injection.The technique is based on the intuition that injection codes implicitly perform a different meaning from general queries. An elaborate environment based on XML for distinguishing between legitimate and malicious users has been presented. Here, the main idea is to secure the database from external users/attackers. Also this method helps us to achieve the same by allowing the web server to access the database only if both the levels of authentication have been satisfied. This is the unique functionality of this proposed method. And also there is no necessity to modify/update the legacy application code, as XML can be easily integrated into other languages. There are other various ways of detecting injection attacks and this is just one of them.

Further, the same method can be extended by adding different levels of authentication within the same application. Hidden web crawling, Syntactic and Semantic Analysis for Automated Testing (Sania), Query Tokenization are the other kinds of methods available.

7. REFERENCES

[1] Mehdi Kiani, Andrew Clark and George , "Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks".The Third International Conference on Availability, Reliability and Security,0-7695-3102-4/08, 2008 IEEE.

[2] V.Shanmughaneethi, C.Emilin Shyni and Dr.S.Swamynathan, "SBSQLID: Securing Web Applications with Service Based SQL Injection Detection" 2009 International Conference on Advances

in Computing, Control, and Telecommunication Technologies, 978-0-7695-3915-7/09, 2009 IEEE

[3] R. Ezumalai, G. Aghila, "Combinatorial Approach for Preventing SQL Injection Attacks", 2009 IEEE International Advance Computing Conference (IACC 2009) Patiala, India, 6-7 March 2009.

[4] Yuji Kosuga, Kenji Kono, Miyuki Hanaoka, Hiyoshi Kohoku-ku, Yokohama, Miho Hishiyama, Yu Takahama, Kaigan Minato-ku, "Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection" 23rd Annual Computer Security Applications Conference, 2007, 1063-9527/07, 2007 IEEE

[5] Ke Wei, M. Muthuprasanna, Suraj Kothari, "Preventing SQL Injection Attacks in Stored Procedures".Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06).[6] NTAGW ABIRA Lambert, KANG Song Lin, "Use of Query Tokenization to detect and prevent SQL Injection Attacks", 978-1-4244-5540-9/10/2010 IEEE.

[7] Prof (Dr.) Sushila, Madan Supriya Madan, "Shielding Against SQL Injection Attacks Using ADMIRE Model", 2009 First International Conference on Computational Intelligence, Communication Systems and Networks, 978-0-7695-3743-6/09 2009 IEEE

[8] A S Yeole, B B Meshram, "Analysis of Different Technique for Detection of SQL Injection", International Conference and Workshop on Emerging Trends in Technology (ICWET 2011) – TCET, Mumbai, India, *ICWET'11*, February 25–26, 2011, Mumbai, Maharashtra, India. 2011 ACM.

[9] Kai-Xiang Zhang, Chia-Jun Lin, Shih-Jen Chen, Yanling Hwang, Hao-Lun Huang, and Fu-Hau Hsu, "TransSQL: A Translation and Validation-based Solution for SQL-Injection Attacks", First International Conference on Robot, Vision and Signal Processing, IEEE, 2011.