SQL Injection and Prevention System

Abstract:

Security and privacy of database-driven web applications are extremely multifaceted against web intruders. One of the most dangerous cyber-attacks is the SQL-injection attack, which simply creates huge loss to commercial vendors. Research deliberates to provide SQL-injection free (SQL-IF) secure algorithm to detect and prevent SQL-injection attacks (SQLIAs). We have re-addressed several detection methods to conflict against the proposed SQL-IF secure algorithm.

Introduction:

Software has pervaded all over the world from past two decades and faced many fascinating challenges. Web applications have become obligatory in human’s day-to-day life while some of the frequently used functional web applications such as online banking, web mail, online auctions, online sales retails, social networks and blogs are the foremost targeted spots for the human web attackers. Web vulnerabilities have made tremendous growth in web applications whereas the web developers fail to meet global standards of designing framework and writing programming code. It is necessary to perform proper input sanitization, syntax validation and follow the security guidelines to secure for prevention of the major loopholes during the programming phase. Many commercial and open source tools exist in market with specialized features even though researchers have analysed and proved not even a single detection scanner provides best result for all the categories of vulnerability. It is highly challengeable task for security-oriented developers to build reliable tools that provide easier approach to handle the security issues. Vulnerability detection scanners are highly intense, used most often among large organizations as they not detect potential vulnerability. Some scanners are not detecting stored vulnerabilities and others are very particular to detect Cross Site Scripting (XSS), Cross Channel Scripting (XCS), Information Leakage, etc. even with some major limitations. The research concentrates by taking SQL injection vulnerability which is one of the common critical attacks on web applications. An existing study performed in 2007 shows the result of about 70% of database-centric web applications are under high risk being hacked by the attackers; eventually most of them are SQL injection attacks.

**1.1 Background of the SQL Injection Attacks (SQLIAs)**: SQL injection attack (code injection) is the most common and easiest type of vulnerability technique adopted by the web attackers through data-driven web applications. By using simple SQL commands such as Select, Where, Insert, Delete and Update, the malicious attackers efficiently re-structure the actual SQL code (statements) and executes vulnerable code into the web applications. Once nasty attacker attain their goals they can easily access sensitive information, modify secured data, executes the data, and even they may collapse the entire application. Since the privacy of the database administrator loses their role by unauthorized accesses of malicious. SQL injection attacks are more lucrative for attackers as they mainly focus to stolen bank account, credit card numbers, etc. This type of security issues on web applications is more susceptible, can be handled by the authentication of users. Many forms of SQL injection attacks exist. Most common takes the benefits of passed parameters, type handling, use of SQL statements Various types of SQL injection attacks are available such as tautologies, illegal/logically incorrect queries, UNION query, Piggy-backed queries, Stored Procedures, Blind SQL, Timing Attack, Alternate Encoding and etc. Defeating these types of attacks is not simple since the attacker actually changes the behaviour of predefined SQL queries. **Methods to Detect and Prevent SQLIAs:** Many research authors explored a number of methods to detect and prevent SQLIAs; the most chosen techniques are static analysis, dynamic analysis, combined static and dynamic analysis, web framework, defensive programming and machine learning techniques. The method of static analysis is extreme were it analyses the code for vulnerability by without actually executing the code. Software metrics and reverse engineering are some forms of static analysis. Model checking, data flow analysis, abstract interpretation and use of assertions in source code are the several techniques of static code analysis. The method of dynamic analysis can be performed automatically by the analysis of vulnerabilities during the execution of web applications which avoids thousands of tests by doing several times manually. Example: CANDID tool. Both the techniques have merits and demerits and therefore variations are identified from the efficacy. However, the research study analysed with various existing works and it has been proved dynamic analysis (penetration testing) tool is effective to test the web applications. Penetration testing tools are easy to use and assure to provide security information systems to their users by fixing the security weaknesses before they get exposed. The major advantages of penetration (dynamic) testing are:

(a) Not necessary to change the development lifecycle

(b) Avoids static analysis challenges

(c) No need for the source code

(d) Deployment-security.

The method of combined static and dynamic analysis can compensate the limitations of each method, which is considered as highly proficient against SQLIAs but it is very complicated. One of the best examples for such a method is AMNESIA tool. It uses static analysis to analyse the web-application code and automatically build a model of the legitimate queries that the application can generate. At runtime, the technique monitors all dynamically-generated queries and checks them for compliance with statically-generated model. When the technique detects a query that violates the model, it classifies the query as an attack, prevents it from accessing the database, and logs the attack information. The web framework method is a filtering method of user input parameters. This method is proven to be in-effective while it is not able filter some special characters. The machine-learning method is the most commonly used method whereas the method results in high false positives and low detection rate

**Literature Survey:**

There are a number of existing tools available, both hardware and software based, to deal with SQL-Injection attacks. Tools exist to detect SQL-Injection attacks while others try to identify and fix SQL-Injection vulnerabilities.

• Green SQL

Green SQL is a free Open Source database firewall that sits between the web server and the database server and is used to protect databases from SQL injection attacks. The logic is based on evaluation of SQL commands using a risk scoring matrix as well as blocking known database administrative commands (e.g., DROP, CREATE, etc). Reports are generated on timestamp, query pattern, reason blocked (e.g., true expression, has 'or' token). It has a white list of approved SQL patterns. However, only MySQL database is currently supported. In comparison, the IDPS in this project may be used with any relational database, not just MySQL. The IDPS has both black and white list pattern features.

• dotDefender

Applicure’s dotDefender is a web application firewall that offers a SQL-Injection solution. dotDefender is a multi-platform solution running on Apache and IIS web servers. Central management ensures a single point of control and reporting for all servers. There is an application layer firewall in front of web applications. It has a set of security rules that enable it to be a powerful solution. However, the cost is prohibitive.

• Code Scan Labs: SQL-Injection

Another product is CodeScan Labs’ SQL-Injection detection product. It has the capability to scan web application source code that you selected for code syntax vulnerabilities. It subsequently generates a "debug style" report. The speed depends on how large the web application is and its complexity.

Current Problems:

Over 15 years after it was first publicly disclosed, SQL injection is still the number one threat to websites. When an IT department noticed an enormous spike in queries to its website and corresponding error messages, they correctly suspected it was the subject of an SQL injection attack. In such case of attack, an attacker sends intentionally malformed requests to a company’s website hoping that the server will malfunction and either return non-public data in response to the request or grant the attacker administrative access to the server. Any (requires interaction with a SQL database) SQL Injection

-Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with has become a common issue with database driven web sites and web applications. Essentially the attack is accomplished by placing a meta character into data input to then place SQL commands in the control plane, which did not exist there before. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

-Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password vulnerabilities.

- Authorization: If authorization information is held in a SQL database, it is very much possible to change this information through the successful exploitation of SQL Injection vulnerability.

- Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection

Proposed System:

The system discussed is called the Intrusion Detection and Prevention System (IDPS). The particular system discussed here is an extension of a particular system that protects a web application system from CGI attacks. However, the original system did not guard against SQL Injection attacks directed at databases connected to the system. We discuss how the SQL-I extension of the Intrusion Detection and Prevention System works in more detail.

**IDPS Detection Models**

There are two models of detection used by this system.

• Signature-based Detection Model

• Anomaly-based Detection Model

**Signature-based (pattern) Detection Model**

In the signature-based detection model, the input obtained from an HTML form is compared to known SQL-I attack patterns (or signatures). If the input is found to match a signature, access is denied and the user is given a generic invalid username/password screen. We intentionally avoid returning a page with an HTTP response or status code which will describe the error that occurred to the user. This is to limit the information and feedback the IDPS system gives to would-be intruders. Even information that seems perfectly harmless can unwittingly give hints about how our system works to attackers which may help them to find a way to circumvent the system’s protections. If a user submits input that matches a known signature an arbitrary number of times, the user’s IP is automatically blocked from accessing the system altogether. An administrator would have to unblock the IP in order for this user to regain access. The signatures themselves would have to be efficient, because a database that contains too many signatures or inefficiently-written signatures would result in poor performance. Also, the signatures would have to be chosen carefully, because we would like to minimize the number of false positives returned. The biggest flaw in the signature-based detection model is it cannot detect attacks that are unknown. That is the reason the IDPS relies on the anomaly-based detection model as a complement to the signature-based detection model.

**Anomaly-based (behavioural) Detection Model**

In the anomaly-based detection model, the number of times a user attempts to log into the system, successful or not, is considered. If the attempts from a user exceed a predetermined number, the system will lock out this user’s IP for a period of time. The user may retry after this time has elapsed. It is important that this period and threshold be arbitrary. This allows the system administrator to determine what the appropriate values for each particular application are, since different systems have different requirements. Anytime the system detects a possible attack, it makes a record of this attack and may block an attacker from accessing the system any further. Furthermore, an alert may be sent to the system administrator. While the IDPS is scanning for attacks, it makes a log of access attempts into the system. This is critical, because if the system administrator wishes to determine if an attack is being attempted at a later time, he has the option of looking back at the access logs to see what input a suspected attack attempted to issue. The system administrator may subsequently block the user if he determines the user was launching an attack on the system. Furthermore, the system administrator or an analyst may examine the log to learn what strategies and patterns the attacker used. This invaluable information may be evaluated and used to develop new SQL-I attack signatures or tweak existing ones.

Implementation:

The IDPS system uses both signature-based and anomaly-based detection models

to identify threats and attacks on the system. Signatures are carefully selected to

implement the signature-based model. Anomaly-based detection is based on the number

of times a user attempted to access the system, regardless of whether any SQL-I patterns

have been detected. IDPS is case-insensitive while trying to use the signature-based

detection method to detect SQL-I attacks. The IDPS deals with White Space Manipulation

attack by removing any white space before comparing text with known SQL-I attack

patterns. The IDPS deals with Comments attack by looking for comment characters in the

submitted text. The String Concatenation attack is dealt with by looking for the

concatenation operation characters or CONCAT function. The keyword “UNION” is searched

for by the IDPS in case an attack tries to perform the UNION Injection attack. The system

will also look for binary, hexadecimal, and decimal characters in the submitted text to catch

instances of this SQL-I attack variation. Sample patterns may be found in the

SQLI\_PATTERNS table described in the IDPS database schema.

Even when no SQL-I attack pattern is detected in the submitted form text, the

IDPS monitors the frequency of the login attempts to implement the anomaly-based

detection method. When the number of visits has exceeded a predetermined threshold,

the system automatically blocks the visitor for a time.

It is significant to note that the screen the user sees when he or she has entered an

incorrect password or when an SQL-I pattern is detected in the text matches. No

feedback is communicated to the user as to whether or not the system has detected an

attack to ensure the system limits unnecessary information broadcast.

Conclusion:

The proposed generic algorithm is substantial in scrutiny of its simple detection mechanism against SQL injection attacks. Testing of web applications for SQL injection attack is a significant step for ensuring its performance and quality. The proposed algorithm performs much faster and endowed with proficient solution to resolve against SQL injection attacks. The paper work has analysed with various detection methods and the proposed method cannot only be implemented on web applications also can be used on any applications which interacts towards databases.

**Future Work:**

The future research will be considerate to construct SQL parser. Generation of parser to detect critical vulnerabilities is another one complex approach. Also dynamic checking complier can be designed to harden the web applications in three-tier internet services for protecting from SQL Injection attacks (SQLIAs). Both the approaches were quite feasible to achieve effectiveness and efficiency

References:

1. Stephen Thomas, Laurie Williams, Tao Xie, on automated prepared statement generation to remove SQL injection vulnerabilities, Journal of Information and Software Technology, Elsevier Ltd, 2009

2. Abdul Bashah Mat Ali , Ala’ Yaseen Ibrahim Shakhatrehb, Mohd Syazwan Abdullahc, Jasem Alostadd, SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks, Journal of Procedia Computer Science, Elsevier Ltd, 2010

3. Joa˜o Antunes, Nuno Neves, Miguel Correia, Paulo Verissimo, and Rui Neves, Vulnerability Discovery with Attack Injection, IEEE Transactions on Software Engineering, 2010, Vol. 36

4. Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon, A novel method for SQL injection attack detection based on removing SQL query attribute values, Journal of Mathematical and Computer Modeling, Elsevier Ltd, 2011

5. Shaukat Ali, Azhar Rauf, Huma Javed, SQLIPA: An Authentication Mechanism against SQL Injection, European Journal of Scientific Research, 2009, Vol.38

6. J. Park, B. Noh, SQL injection attack detection: profiling of web application parameter using the sequence pairwise alignment, Journal of Information Security Applications, LNCS, 2007, vol. 4298

7. http://www.ijarcs.info/index.php/Ijarcs/article/viewFile/3076/3059

8. Webpage “Wikipedia.org: SQL”

http://en.wikipedia.org/wiki/SQL

9. Webpage “Homepage for GreenSQL”

http://www.greensql.net/ Retrieved on 2009-12-15.

10. Webpage “About page for dotDefender from Applicure”

http://www.applicure.com/About\_dotDefender Retrieved on 2009-12-15.

11. Webpage “About page for CodeScan from CodeScan Limited”

http://www.codescan.com/about-codescan/what Retrieved on 2010-04-10.

12. Aulakh, T. Intrusion Detection and Prevention System: CGI Attacks, 2009. San

Jose State University master’s thesis project.