Volume : 2 | Issue : 6 | June 2013 • ISSN No 2277 - 8179

Research Paper

Security of Database Query Processing by Blocking SQL Injection Attacks

Engineering

KEYWORDS : Database Security, SQL Injection, Link

Rahul Pancholi	Department of Computer Engineering, Hasmukh Goswami College of Engineering, Vehlal, Gujarat
Indr jeet Rajput	Department of Computer Engineering, Hasmukh Goswami College of Engineering, Vehlal, Gujarat
Vinit Kumar	Department of Computer Engineering, Hasmukh Goswami College of Engineering, Vehlal, Gujarat

ABSTRACT

Online systems are made up of some infrastructure components and application specific code. The major problem with this kind of system is hacker compromises the system and enters into back end of the database system. The SQLIA(SQL Injection attacks are suck kind of attacks that are highly susceptible or vulnerable. There is a lack of some robust defense against suck SQLIA. To prevent such problems of the existing system is link representation of a valid query. Which will generate and store the valid structure of queries in the ordered sequence of tokens. To overcome processing overhead and fast searching multithreading is used. Another way to overcome huge computation overhead valid fingerprint of generated token will be stored as integer value. And that integer value will be searched instead of the string or query from the specific group. This technique will prevent SQLIA without source code modification and will provide higher level efficiency

1. INTRODUCTION

The Detection and prevention of SQLIAs is a major issue now a days in industry. To fulfil this task many tools and techniques have been introduced but none of them are capable enough to provide a fool proof mechanism that can assure the absolute and higher level of security on web based application system. The survey and research on this reveals that there is a lack of complete mechanism or we can say methodology as most of their requires source code modification. So there is a requirement of a technique or mechanism which can be easily deployable and should not require the source code modification and must assure the performance and security. The most important asset in any business or other organization is Information. But SQL Injection Attacks (SQLIAs) violates it. In this competitive edge SQLIAs are one of the top level threats for such kind of organization and web applications. SQLUA are the attack which lies in the most serious vulnerability types. The survey on SQLIA in 2008 states that such attacks are increased by 134% [2] and it has become prime vulnerability issue. Web applications are easy to detect and exploit by the external hacker ; and that's why the SQLIAs are habitually used by malevolent users for different reasons like Financial fraud, theft of confidential data, deface website, harm, spying, cyber terrorism, or simply for fun. And SQLIA methods have become more common and sophisticated; so effective and feasible solution for this problem in the sensitive field of data, information and computer security is deeply needed. To achieve this goal, so many solution like automatic tools and some systems of security have been implemented. But the problem is none of them which have been implemented cant guarantee an high and absolute level of security on web based application. So here is the mechanism which is easily deployable, there is no need of source code modification. In addition to that it also provide a good performance and overall efficiency.

2. Previous Work

2.1 Framework Support:

Now a days Recent frameworks for web applications provides a mechanism that can prevents the SQL injections. An input validator prohibits user input which is provided by user, from including meta-characters to avoid SQL injections. But if we or any user want to include meta- characters in the input which is needed, we can not prevent SQL injections.

2.2 Prepare Statement [1]:

SQL provides the functionality which is prepared statement, the working of it separates the values in a query provided from the structure of SQL. The programmer defines a valid skeleton of an SQL query and then fills in the holes of the skeleton at runtime. The prepare statement makes it harder to inject SQL queries as the SQL structure can not be altered. But To use the prepare statement functionality, we are suppose to modify the web application entirely; all the legacy web applications must be rewritten to shrink the likelihood of SQL injections.

2.3 Static Analysis [2]:

Wassermann and Su projected an approach that uses a static analysis in combination with automated reasoning. This technique verifies the tautological attacks but other types of attacks cant be prevented by this static analysis technique.

2.4 Machine Learning Approach [3]:

Valeur et al. projected the use of an intrusion detection system (IDS) based on a machine learning technique. IDS is trained by expert using a set of typical application queries and ultimately builds models of the typical queries, and then monitors the application at runtime to find out the queries that do not match the designed model. The overall IDS quality and efficiency depends on the quality of the training set; a poor training set and analysis would result in a large number of false positives and negatives.

2.5 Instruction-Set Randomization [4]:

SQLrand provides a framework that allows developers to create SQL queries using randomized keywords instead of the normal SQL keywords. A proxy between the web application and the database intercepts SQL queries and de-randomizes the keywords. The input provided by an attacker will not be able to construct the randomized keyword and the input command would result into a wrong syntax and it will be proved syntactically incorrect query. SQLrand uses a secret key to modify keywords, its security relies on attackers not being able to discover this key. SQLrand requires the application developer to rewrite code.

2.6 Taint-Based Technique [5]:

Pietraszek and Berghe modified a PHP interpreter to track taint information at the character level. The taint based technique uses a context-sensitive analysis and rejects SQL queries if an un-trusted input has been used to create certain types of SQL tokens. A common weakness of this approach is that they require modifications to the runtime environment, which reduces the portability.

2.7 Combined Static and Dynamic Analysis:

Su et al. [8] present grammar-based approach to detect and stop queries having SQLIAs by implementing SQLCHECK tool. They mark user supplied portions in queries with a special symbol and augment the standard SQL grammar with production rule. A parser is generated based on the augmented grammar. The

Volume : 2 | Issue : 6 | June 2013 • ISSN No 2277 - 8179

parser successfully parses the generated query at runtime, if there are no SQLIAs in the generated queries after adding user inputs. This approach uses a secret key to discover user inputs in the SQL queries. Thus, the security of the approach relies on attackers not being able to discover the key. Additionally, this approach requires the application developer to rewrite code to manually insert the secret keys into dynamically generated SQL queries.

3.WEB APPLICATIONS

Web applications are everywhere in our daily internet use. Common examples are those applications used for searching the internet such as "Google"; for collaborative open source projects as "SourceForge"; for public auctions as "eBay" and many others as well as blogs, webmail, web-forums, shopping carts, e-commerce, dynamic contents, discussion boards and social networks. At the moment, according to survey conducted by cnet.com Gmail, Amazon, yahoo, live search are coming under the top 100 web application [8].The core part of a web application, as stated above, is stored on the server-site within the application server. This core consists of a real computer software program coded in a browser supported programming language such as PHP, ASP, CGI, Perl, Java/JSP, J2EE.

3.1 Input Validation-Based Vulnerabilities

To highlight how everywhere web applications have become and how prevalent their problems are, for each year from 2001 to 2006, the percentage of newly reported security vulnerabilities in five vulnerability classes (this data comes from Mitre‟s report [8]): All of these except buffer overflows are specific to web applications.SQL injections are consistently at or near the top: 13% of the reported vulnerabilities in 2006

3.2 Most Attacked Organizations

The largest category of hacked organizations is government and related organizations (Law Enforcement and Politics). Combine those categories with education in 6th place and it appears that the non-commercial sector represents the primary target for hackers (this data comes from the Web Hacking Incidents Database)[9]. Government is a prime target due to ideological reasons, while universities are more open than other organizations. These statistics, however, are biased, to a degree, as the public disclosure requirements of government and other public organizations are much broader than those of commercial organizations

4.SQLIA RECENT SCENARIO

In 2008, SQL injection replaced cross-site scripting as the predominant Web application vulnerability. The overall increase of 2008 Web application vulnerabilities can be attributed to a huge spike in SQL injection vulnerabilities, which was up a staggering 134% from 2007 [10]. In the first half of 2008, it has been seen that instead of leveraging SQL injection to steal data, this attack updated the application's back-end data to include iFrames to redirect visitors to malicious Web pages[10]. These attacks targeted many well-known as well as many trusted Web sites also.

5. SQL INJECTION

SQL injection is a technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed. It is in fact an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.

5.1 How SQL Injection Attacks (SQLIAs) Work

SQL injection refers to a class of code-injection attacks in which data provided by the user is included in the SQL query in such a way that part of the user's input is treated as SQL code. It is a trick to inject SQL query or command as an input possibly via the web pages. They occur when data provided by user is not properly validated and is included directly in a SQL query. By leveraging these vulnerabilities, an attacker can submit SQL commands directly access to the database. There are two major SQL Injection techniques:

i) access through login page or user input and
ii) ii) access through URL.

5.2 The first technique is the easiest in which it bypasses the login forms where users are authenticated by using passwords. This kind of technique can be performed by the attackers through: „or‟ condition, „having‟ clause, multiple queries and extended stored procedure. This kind of vulnerability represents serious threats.

SELECT * FROM users WHERE username = ' " & userName & " ' AND password = ' "

& userPass & " ' "

If the username and password as provided by the user are used, the query to be submitted to the database takes the form;

SELECT * FROM users WHERE username = 'guest' AND password = 'guestpass'

If the user were to enter [„ OR 1=1 --] and [ ] instead of [guest] and [guestpass], the query would take the form;

SELECT * FROM users WHERE username = ' ' OR 1=1 –' AND password = ' '

The query now checks for the conditional equation of [1=1] or an empty password, then a valid row has been found in the users table. The first [„] quote is used to terminate the string and the characters [--] mark the beginning of a SQL comment, and anything beyond is ignored. The query as interpreted by the database now has a tautology and is always satisfied. Thus an attacker can bypass all authentication modules gaining unrestricted access to critical information on the server. SQL injection potentially affects every database on all platform and web application. This attack can be used to gain confidential information, to bypass authentication mechanisms, to modify the database, and to execute arbitrary code.

5.3 The second technique can be performed by the attackers through:

manipulating the query string in URL and using the SELECT and UNION statements.

When a user enters the following URL:

http://www.mydomain.com/products/products.asp?productid=123

The corresponding SQL query is executed:

SELECT ProductName, ProductDescription FROM Products WHERE ProductNumber = 123

An attacker may abuse the fact that the ProductID parameter is passed to the database without sufficient validation. The attacker can manipulate the parameter's value to build malicious SQL statements. For example, setting the value [123 OR 1=1] to the ProductID variable results in the following URL:

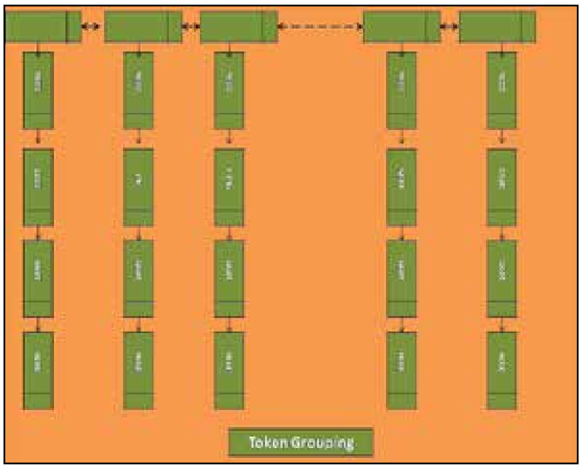
http://www.mydomain.com/products/products.asp?productid=123 or 1=1

The corresponding SQL Statement is:
SELECT ProductName, Product Description From Products WHERE ProductNumber = 123 OR 1=1

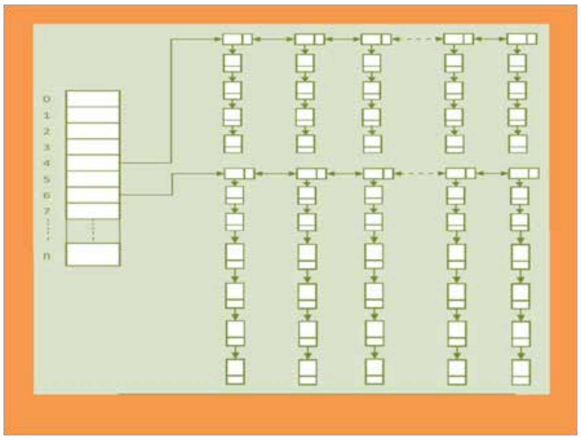
6.PROPOSED TECHNIQUE

1. Take SQL query as an input which is having valid structure.E.g. Select * from customer where city=kolkata;
2. Sequence of token after token separation. 2236 2377 1099 3970 1591 114 61 4074

3. Linked Representation of translated tokens. 2236->2377->1099->3970->1591->114->61->4074
4. Grouping of tokens having same number of tokens.



5. **Complete structure of all valid queries.**



Future Work

Security module can only processes a single SQL query it cannot handle PL/SQL code blocks so in future we expand our solution to handle PL/SQL code block. As there is a limitation of this technique that it cannot detect such type of attack caused by only changing the data value of a query but not changing the structure of the query so we will try to improve our solution.

ACKNOWLEDGMENT

I would be thankful to my guide HOD Mr Indr Jeet Rajput and assistant professor Mr Vinit kumar Gupta here for dedicated help when I have some troubles in writing. I will also thank my class mates in laboratory for their concern and support both in study and life

REFERENCE

[1] Stephen Thomas, Laurie Williams. Using Automated Fix Generation to Secure SQL Statements. Third International Workshop on Software Engineering for Secure Systems (SESS'07), pages 9-9,May 2007. | [2] G. Wassermann and Z. Su. An Analysis Framework for Security in Web Applications. In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS), pages 70–78, 2004. | [3] F. Valeur, 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 and Vulnerability Assessment (DIMVA), pages 123–140, 2005. | [4] S. Boyd and A. Keromytis. SQLrand: Preventing SQL injection attacks. In Proceedings of the Applied Cryptography and Network Security (ACNS), pages 292–304, 2004. | [5] T. Pietraszek and C. Vanden Berghe. Defending against Injection Attacks through Context-Sensitive String Evaluation. In Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID), pages 124–145, 2005. | [6] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. Annual Symposium on Principles of Programming Languages (POPL), pages 372–382, 2006. | [7] CNET.com. 2009 Webware 100 winners. 2009. http://www.webware.com/100/ | [8] Steve Christey. Vulnerability Type Distributions in CVE. cwe.mitre.org,Oct2006. http://cwe.mitre.org/documents/vuln-trends.html. | [9] Breach .com, The Web Hacking Incidents Database Annual Report 2009. Breach Security Whitepaper, Feb 2009. | http://www.breach.com/resources/whitepapers/downloads/WP_WebHackingIncidents | _2008.pdf | [10] IBM, IBM Internet Security SystemsX-Force 2008 Trend & Risk Report, Jan 2009, | http://www935.ibm.com/services/us/iss/xforce/trendreports/xforce-2008-annual-report.pdfs |