International Journal of Technology and Engineering System (IJTES)
Vol 7. No.3 2015 Pp. 219-223

available at : www.ijcns.com
ISSN: 0976-1345

REVIEW OF SQL INJECTION DEFENSE TECHNIQUE BASED ON THE ANALYSIS OF DIFFERENT WEB BASED ATTACKS

RANI S.K, S.MOHAMED SALEEM
[1]*PG Scholar,* [2]*PG Scholar*
Department of Computer Science and Engineering
Valliammai Engineering College
Chennai, India
[1]rsk1411@gmail.com, [2]mohamedsaleem25@gmail.com

ABSTRACT

Over the last few years, hacker community constantly researched about new techniques to exploit web applications in a sophisticated manner and bypass traditional security controls. More troubling is that manipulating the application, the hacker can use the trusted relationship between the application server and the database to gain inappropriate access to sensitive information found on the site. Knowledge of various attack techniques and their characteristic is significant to protect both the application and the database. In this paper focuses on different types of SQL injection attack, its detection and prevention techniques. This study helps to protect your business from the earlier stages of the attack with respect to the nature of the environment it takes place.

Keywords—Database Security; Information Security; Web attacks; SQL Injection;

I .INTRODUCTION

Web based attacks [1] are considered by security expertsto be the greatest and oftentimes the least understood of all risks related to confidentiality, availability, and integrity. Application vulnerabilities could provide the means for malicious end users to breach a system's protection mechanisms typically to take advantage or gain access to private information or system resources. Information gathered can include social security numbers, dates of birth, and maiden names, which are all often used in For example: numerous downloader Trojans use the web as a simple file repository, downloading other malicious files via HTTP. Malicious scripts hosted on attack sites await the visit of vulnerable client browsers before they unleash exploit code in order to infect the victim. This paper focus on analysis of different web based attacks and techniques for detection and prevention of SQL injection attack [2].

A. Anatomy of a Web attack

1. Attacker breaks into a legitimate Web site and posts malware. Malware is no longer exclusive to malicious identity theft. Another popular target for attackers is credit card data which left unprotected and unencrypted can be used to cause significant damage to organizations most valued assets, their customers. Over the past two years, malware has evolved to make increased use of the web. The scope extends further than just malicious scripts embedded in web pages, mainstream Web sites to act as parasitic hosts that serve up Websites. Today it is common place for legitimate malware to their unsuspecting visitors.

Fig.1 Anatomy of Web attacks

2. Attacking end user machines Malware on a Web site makes its way down on to a user's machine when that user visits the host Web site.

3. Leveraging end user machines for malicious activity.The most malicious activities begin once new malware has established a presence on a user's machine.

B. SQL Injection Attack

One popular type of attack involves compromising the database using a technique known as SQL injection.

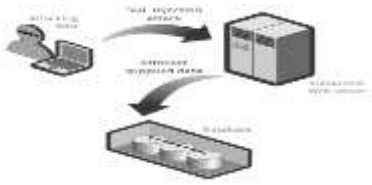

Fig.2 SQL Injection Attack

II .TYPES OF SQL INJECTION ATTACKS

1. Ilegal/Logically Incorrect Queries: when a query is rejected, an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical errors by purpose. In this example attacker

http://www.arch.polimi.it/eventi/?id_nav=8864

2)SQL Injection:

http://www.arch.polimi.it/eventi/?id_nav=8864'

3) Error message showed:

SELECT name FROM Employee WHERE id =8864\' From the message error we can find out name of table and fields: name; Employee; id. By the gained information attacker can organize more strict attacks.

2. Piggy-backed Query: Extracting data, adding or modifying data, performing denial of service, executing remote commands. In the piggy-backed Query attacker tries to append additional queries to the original query string. On the successful attack the database receives and executes a query string that contains multiple distinct queries. In this method the first query is original whereas the subsequent queries are injected. This attack is very dangerous; attacker can use it to inject virtually any type of SQL command. For example, SELECT * FROM user WHERE id='admin' AND password='1234'; DROP TABLE user; --'; Here database treats above query string as two query separated by ";" and executes both. The second sub query is malicious query and it causes the database to drop the user table in the database.

3. Blind SQL Injection: In this type of attack, useful information for exploiting the backend database is collected by inferring from the replies of the page after questioning the server some true/false questions. It is very similar to a normal SQL Injection. However, when an attacker attempts to exploit an application, rather than getting a useful error message, they get a generic page specified by the developer instead. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker can still get access to sensitive data by asking a series of True and False questions through SQL statements.

Scenario

http://victim/listproducts.asp?cat=books

SELECT * from PRODUCTS WHERE category='books'

http://victim/listproducts.asp?cat=books' or '1'='1.

SELECT * from PRODUCTS WHERE category='books' or '1'='1'.

4. Union Query

Union is a command in the database which work for combining two queries, by using UNION command, attacker merges their specially crafted queries with the original query to extract the records from a table other than the one intended by the developer. Continuing with our running example, an attacker can invoke an union-query attack using the URL

http://localhost/?EmpId=' UNION<SQL statement>.

This url will render the following SQL statement.

SELECT empinfo FROM EmpTable WHERE EmpID = " UNION <SQL statement>. The second SQL statement will get executed.

The signature of union-query attack is the UNION meta character of SQL.

5. Alternate Encodings

The main goal is to escape detection. In this attack, the injected text is modified so as to avoid detection by defensive coding practices and also many programmed prevention techniques. This type of attack is used in unification with other attacks. In other words, substitute encodings do not provide any single 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 replaceable. These elusion techniques are often necessary because a common defensive coding practice is to scan for certain known "corrupt characters" such as single quotes and comment operators

Example: $login = mysql_query("SELECT * FROM user WHERE (User_ID= '" . mysql_real_escape_string($_POST['User_ID']) . "') and (pass_word = '" . mysql_real_escape_string($_POST['pass_word']) . "')");

To evade this defence, attackers have employed alternate methods of encoding their attack strings (e.g., using hexadecimal, ASCII, and Unicode character encoding). Common scanning and detection techniques do not try to evaluate all specially encoded strings, thus allowing these attacks to go undetected.

6. Timing Attacks: A timing attack lets an attacker gather information from a database by observing timing delays in the database's responses. This technique by using if-then statement cause the SQL engine to execute a long running query or a time delay statement depending on the logic

injected. This attack is similar to blind injection and attacker can then measure the time the page takes to load to determine if the injected statement is true. This technique uses an if-then statement for injecting queries. WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time.

For example, in the following query: declare @s varchar(8000) select @s = db_name() if (ascii(substring(@s, 1, 1)) & (power(2, 0))) > 0 waitfor delay '0:0:5'

Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1. Then code is then injected to generate a delay in response time when the condition is true. Also, attacker can ask a series of other questions about this character. As these examples show, the information is extracted from the database using a vulnerable parameter.

7. Stored Procedure : Performing privilege escalation, performing denial of service, executing remote commands. Description: In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. Stored procedure is nothing but a code and it can be vulnerable as program code. For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input "; SHUTDOWN; --" for username or password. Then the stored procedure generates the following query: For example, SELECT accounts FROM users WHERE login= '1111' AND pass='1234 '; SHUTDOWN;--; This type of attack works as piggyback attack

III . DETECTION AND PREVENTION TECHNIQUES

SQLrand Scheme - SQLrand approach [8] is proposed by Boyd and Keromytis. For the implementation, they use a proof of concept proxy server in between the Web server (client) and SQL server; they de-randomized queries received from the client and sent the request to the server. This de-randomization framework has 2 main advantages: portability and security. The proposed scheme has good performance: 6.5 ms is the maximum latency overhead imposed on every query.

1. Automated Approaches - Besides using manual approaches, [9] also highlights the use of automated approaches. The author notes that the two main schemes are: Static analysis FindBugs and Web vulnerabilityscanning. Static analysis FindBugs approach detects bugs on SQLIAs, gives warning when an SQL query is made of variable. However, for Web vulnerability scanning, it uses software agents to crawl and scans Web applications and detects the vulnerabilities by observing their behavior.
2. Parse Tree Validation Approach - Buehrer et al. adopt the parse tree framework. They compared the parse tree of a particular statement at runtime and its original statement. They stopped the execution of statement unless there is a match. This method was tested on a student Web application using SQLGuard. Although this approach is efficient, it has two major drawbacks: additional overheard computation and listing of input (black or white).
3. Swaddler- Swaddler [10] analyzes the internal state of a web application. It works based on both single and multiple variables and shows an impressive way against complex attacks to web applications. First the approach describes the normal values for the application's state variables in critical points of the application's components. Then, during the detection phase, it monitors the application's execution to identify abnormal states.
4. AMNESIA - In [11], Junjin proposes AMNESIA approach for tracing SQL input flow and generating attack input, JCrasher for generating test cases, and SQLInjectionGen for identifying hotspots. The experiment was conducted on two Web applications running onMySQL1 1 v5.0.21. Based on three attempts on the two databases, SQLInjectionGen was found to give only two false negatives in one attempt. The proposed framework is efficient considering the fact that it emphasizes on attack input precision. Besides that, the attack input is properly matched with method arguments. The only disadvantage of this approach is that it involves a number of steps using different tools.
5. Manual Approaches – [12] highlights the use of manual approaches in order to prevent SQLI input manipulation flaws. In manual approaches, defensive programming and code review are applied. In defensive programming: an input filter is implemented to disallow users to input malicious keywords or characters. This is achieved by using white or black lists. As regards to the code review [24], it is a low cost mechanism in detecting bugs however it requires deep knowledge on SQLIAs.
6. Ali et al.'s Scheme - [13] adopts the hash value approach to further improve the user authentication mechanism. They use the user name and password hash values. SQLIPA (SQL Injection Protector for Authentication) prototype was developed in order to test the framework. The user name and password hash values are created and calculated at runtime for the first time the particular user account is created.
7. Haixia and Zhihong's Scheme - In [14], Haixia and Zhihong propose a secure database testing

design for Web applications. They suggest a few things; firstly, detection of potential input points of SQL Injection; secondly, generation of test cases automatically then finally finding the database vulnerability by running the test cases to make a simulation attack to an application. The proposed methodology is shown to be efficient.

8. SecuriFly: SecuriFly [15] is tool that is implemented for java. Despite of other tool, chase string instead of character for taint information and try to sanitize query strings that have been generated using tainted input but unfortunately injection in numeric fields cannot stop by this approach. Difficulty of identifying all sources of user input is the main limitation of this approach.
9. SQL Prevent: SQL Prevent is consists of an HTTP request interceptor. The original data flow is modified when SQL Prevent is deployed into a web server. The HTTP requests are saved into the current thread-local storage. Then, SQL interceptor intercepts the SQL statements that are made by web application and pass them to the SQLIA detector module. Consequently, HTTP request from thread-local storage is fetched and examined to determine whether it contains an SQLIA. The malicious SQL statement would be prevented to be sent to database, if it is suspicious to SQLIA.
10. SQLCHECK: Su and Wassermann implement their algorithm with SQLCHECK on a real time environment. It checks whether the input queries conform to the expected ones defined by the programmer. A secret key is used for the user input delimitation. The analysis of SQLCHECK shows no false positives or false negatives. Also, the overhead runtime rate is very low and can be implemented directly in many other Web applications using different languages.

11. SQL-IDS Approach - Kemalis and Tzouramanis suggest using a novel specification-based methodology for the detection of exploitations of SQL injection vulnerabilities. The proposed query-specific detection allowed the system to perform focused analysis at negligible computational overhead without producing false positives or false negatives.
12. Context Sensitive String Evaluation (CSSE) The basic idea behind this approach is to find out the root cause of SQLIA [31]. The root cause is the origin of the data (information about the data, termed as metadata) i.e., user-provided or developer-provided. Thus, any data provided by the user is marked as untrusted and data provided by the applications are termed as trusted. The untrusted metadata are used for syntactic analysis based on 'Context Sensitive String Evaluation (CSSE)'. Injection may also occur due to programming flaws during developments. CSSE is basically based on syntactical analysis, which first distinguishes string constants (for e.g., select *from users where login='$login_name') and numerical constants (e.g., select * from users where pin=$pin). It then removes all unsafe characters (un-escaped quotes) in alphanumeric identifiers and non-numeric characters in numeric identifiers. This operation is performed before sending the query to the database server. Following issues are there in this approach (i) Initialization of the unsafe characters is dependent on the web programmer, and (ii) Removal of unsafe characters restricts the application functionality.
13. CANDID Bishtet al proposed CANDID. It is a Dynamic Candidate Evaluations method for automatic prevention of SQL Injection attacks. This framework dynamically extracts the query structures from every SQL query location which are intended by the developer (programmer). Hence, it solves the issue of manually modifying the application to create the prepared statements.
14. SAFELI - proposes a Static Analysis Framework in order to detect SQL Injection Vulnerabilities. SAFELI framework aims at identifying the SQL Injection attacks during the compile-time. This static analysis tool has two main advantages. Firstly, it does a White-box Static Analysis and secondly, it uses a Hybrid-Constraint Solver. For the White-box Static Analysis, the proposed approach considers the byte-code and deals mainly with strings. For the Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables

IV . CONCLUSION

In this paper different types of SQL Injection attacks and their defence technique is surveyed. However, there are several medium are available to attempt various web based attacks, the major cause for loss of database security is user unawareness. This study will help the user to get full-fledged knowledge about SQL injection. Advantages and disadvantages of different web based attacks also discussed.

REFERENCES

[1] *http://www.darkreading.com/risk/10-web-based-attacks-targeting- yourend-users/d/d-id/1140224*

[2] *https://www.owasp.org/index.php/SQL_Injection*

[3] *Mohammed Alenezi and Martin J. Reed School of Computer Science and Electronic Engineering "Denial of Service Detection Through*

TCP Congestion Window Analysis" World Congress on Internet Security 2013.

[4] Piromsopa, K. and Enbody, R.J. "Buffer Overflow Protection: The Theory," IEEE International Conference on Electro/information Technology, 2006.

[5] Yongle Wangs Xuchang ploughs the recent information science research institute Xuchang, JunZHang Chen Xuchang Vocational Technical College Xuchang, China"Hijacking spoofing attack and defense strategy based on Internet TCP sessions" 2013

[6] I.Ullah, N.Khan, and H.A Aboalsamh, "Survey on botnet: Its architecture, detection, prevent ion and mitigation" 10th international conference on Network Sensing and Control.

[7] Paul C. Kocher, "Timing Attacks on Implementations of DiffieHellman,RSA, DSS and other systems," cryptography research Inc.

[8] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL Attacks. In Proceedings of the 2nd Applied Cryptography and Network Security Conference, pages 292–302, June 2004.

[9] Mei Junjin, "An Approach for SQL Injection Vulnerability.

[10] Detection," Proc. of ITNG '09, pp.1411-1414, 27-29 April 2009.

[11] Macro Cova, Davide Balzarotti. Swaddler:" An Approach for the Anomaly-based Detection of State Violations in Web Applications", Recent Advances in Intrusion Detection, Proceedings, volume: 4637 Pages: 63-86 Published: 2007

[12] M. Junjin, "An Approach for SQL Injection Vulnerability Detection," Proc. of the 6th Int. Conf. on Information Technology: New Generations, Las Vegas, Nevada, pp. 1411 1414, April 2009.

[13] Mei Junjin, "An Approach for SQL Injection Vulnerability Detection," Proc. of ITNG '09, pp.1411-1414, 27-29 April 2009. Mechanism Against SQL Injection," European Journal of Scientific Research ISSN 1450-216X Vol.38 No.4 (2009), pp 604-611

[14] Y. Haixia, N. Zhihong, "A database security testing scheme of web application," Proc. of ICCSE '09 , pp. 953-955, 2009.

[15] M. Martin, B. Livshits, and M. S. Lam., " Finding Application Errors and Security Flaws Using PQL: A Program Query Language" ACM SIGPLAN Notices, Volume: 40, Issue: 10 Pages: 365-383, 2005.