ISSN: 2277-9655
Impact Factor: 4.116

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

CASE STUDY OF SQL INJECTION ATTACKS

Sonakshi*, Rakesh Kumar, Girdhar Gopal
* Department of Computer Science and Applications, Kurukshetra University Kurukshetra India

DOI: 10.5281/zenodo.56935

ABSTRACT

Today, most of the web applications are associated with database at back-end so there are possibilities of SQL injection attacks (SQLIA) on it. A number of preventive measures have also been discovered by various researchers to overcome this attack, but which measure is more convenient and provides fast access to application without compromising the security is also a major concern nowadays. This paper provides a clear distinction among different types of SQLIAs and how these can be performed on local server. Also, demonstration of SQLIAs on live websites is provided for better understanding of URL attacks. Finally, a complete set of guidelines is provided to help understand the causes of various SQLIAs and how to detect them prior and their preventive measures for the developers of database-driven web applications and researchers.

KEYWORDS:Database-driven web applications, security of web applications, SQL injection, SQLI attack, Vulnerabilities, Web applications attack etc.

INTRODUCTION

These days, the rapid development of web technology, web applications have been widely used in many fields, we depend on internet or web applications for most of the activities such as online banking, shopping from e-commerce sites, social networking, online bill payments and transactions etc. in our daily life. According to Imperva's Community Defense service, out of 300,000 attack campaigns that have occurred globally, 24.6 percent were SQLi attacks in 2014 [1]. Computers and Internet equipped devices have become an integral part of human life. As per the Statistics, number of Internet users in India is the third highest after China and United States. According to OWASP report 2013, SQL Injection tops the list among the top 10 vulnerabilities, shown below for web applications. Web applications are the applications that can be accessed over the internet by using any web browser that runs on any operating system and architecture. They have become ubiquitous due to the flexibility, availability, convenience and interoperability that they provide [2]. These applications operate with sensitive user information and hence there is high need for assuring their confidentiality, integrity and availability. Web site features such as contact forms, login forms, support requests, search functions, feedback fields, shopping carts and even the functions that deliver dynamic web page content, are all susceptible to SQL injection attack because the very fields presented for the visitors must allow use of at least some SQL commands to pass through directly to the database. Database driven web applications have become widely deployed on the internet. Organizations make their use to provide a broad range of services to their customers. These applications and their underlying databases contain confidential or even sensitive information such as credit card numbers, customer or financial records. This information can be highly valuable and makes web applications an ideal target for attackers [3]. SQL injection is a code injection technique used to attack database driven web applications. In this attack, the attacker inserts a portion of SQL statement via not sanitized user input parameters into the original query and passes this to the database server. SQL Injection is the most prevalent form of attack among the website and database attacks which are exploited these days and is being applied on several websites which are not secured properly. There are many techniques available to prevent SQL injections, but still it is a big issue in development area. Because Black hat hackers are always busy to find out new techniques and applying them in new way. The main focus of this paper is to make programmers and researchers aware of SQL injection techniques which are still prevalent these days.

This paper is organized in six sections as follows. Section 2 presents the types of SQL injections. Attacks on local web server are discussed in section 3. Demonstration of attacks on live websites is provided in section 4. Then, a set of guidelines is provided in section 5. Finally, conclusion is provided in section 6.

SQL INJECTION TYPES

The attacker can plan a specific attack according to the particular kind of vulnerability present in the application because these are the flaws or weaknesses present in the system. For a successful SQLIA, the attacker should append a syntactically correct command to the original SQL query. There are various kinds of SQLIA, which will be discussed in this section. The first one is the tautology which is the most common type of SQLIA. In this, the attacker can inject malicious code into more than one conditional query statement to be evaluated always true. The next is the logically incorrect queries in which database generate error messages which often contain useful information that allow attacker to find out the vulnerable parameter in an application and the database schema. The intention of attacker is to gather information about the type and structure of back end database [4]. In case of union attacks, the database returns a dataset that is the union of the results of the original query with the results of the injected query [2]. In case of piggy-backed query attack, the additional query is injected into the original one. As a result, database has to process multiple queries simultaneously. Database treats this query as two different queries which are separated using the delimiter (';'). Normally the first query is legitimate query, whereas the following query could be illegitimate. So, the attacker can inject virtually any type of SQL command to the database [5] [6] [7] [8]. For example, the attacker injects "O; drop table user" into the pin input field instead of logical value. In stored procedure attacks, the attacker tries to execute already present stored procedures in the database. Through this, the attacker identifies the current database being used. To perform this attack, the attacker injects' '; SHUTDOWN; into either name or password, which would result in shut down of the database [9] [10]. In case of Inference attacks, the attacker observes the response of the webpage after injecting some malicious code into it. With this type of attack, intruders change the behavior of a database or application [11]. "Timing Attacks" and "Blind Injections" are the two kinds of inference attacks techniques. In blind injection, the developers hide error details from the attackers, so that they won't get any help from the database. In this case, the attacker faces a generic page provided by developer, instead of an error message. An attacker can still steal the data by asking a series of True/False questions through SQL statements [6] [12]. A timing attack lets an attacker, gather information from a database by observing timing delays in database responses. This technique makes the use of "if-then" statement which causes the SQL engine to execute a long running query or a time delay statement depending upon the logic injected. In time based attacks, attacker introduces a delay by injecting an additional SLEEP(n) call into the query and then observing if the webpage was actually delayed by n seconds [13].

SQLIAs ON LOCAL SERVER

There are many types of SQLIAs and its countless variations. Researchers are often unaware of the myriad of different techniques that can be used to perform SQLIAs. Therefore, most of the solutions proposed, detect or prevent only a subset of the possible SQLIAs. To address this problem, a comprehensive survey of SQL injection attacks known till date, is presented. For each type of attack, a characterization of the attack, its effects are illustrated and examples of how those types of attack could be performed and how these are prevented using our own framework designed using PHP and MySQL. Malicious SQL statements can be introduced into a vulnerable application using many different input techniques. These are explained below.

Case 1: The most basic case in which authorized user interacts with user interface of web based applications using authenticated login credentials.

In this case, the valid user interacts with the login interface of web application using authenticated username and password with which the user has registered. The registered username and password are "sonakshi". The same has been demonstrated using WAMP server 2.2 and PHP as server scripting language on Windows 7 platform. It is shown below in Fig. 1.

ISSN: 2277-9655
Impact Factor: 4.116

Fig. 1 Authenticated username and password

The SQL query will be formed using above authenticated login credentials as:
$sql = "select * from users where username='sonakshi' and password='sonakshi' ";
As the user is authenticated, so after pressing "signin" button, that user is allowed to access his/her account without any restrictions. The next page will be displayed as in Fig. 2.

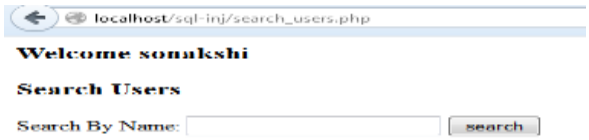

Fig. 2 Redirected page after authorized user login

Case 2: Tautology Based attacks
The main goal of this type of attack is to inject code in one or more conditional statements so that they are always evaluated to be true. The consequences of this type of attack, is dependent upon how the results of the query are used within the application. The attack is successful when the injected code either displays all of the returned records or performs some action if at least one record is returned [3]. E.g. an attacker submits " **'or 1=1 - -** " for the login input field. The resulting query will be formed as:
$sql = "select * from users where username='sonakshi 'or '1=1- - and password='anypassword' "; It is shown in the Fig. 3 below.

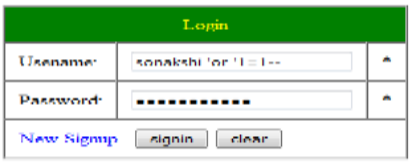

Fig. 3 Tautology attack

After pressing the "signin" button, the attack is successful and attacker is redirected to next page.

Prevention:
Its prevention can be done by using a built in function in PHP called *mysql_real_escape_string()*, which takes a string which will be used in a MySQL query and return the same string with all SQL Injection attempts safely escaped. Basically, it will replace those troublesome quotes ('), entered by user with a MySQL-safe substitute, an escaped quote \'.The *mysql_real_escape_string()* function is used to escape special characters in a string like \x00, \n, \r, \, ', " and \x1a.

Another method which can be used to prevent tautology-based attacks is to define a function manually i.e. user-defined function. Here, a function is defined which is named as *"validate()"* which checks for invalid tokens such as "--","=","or", "","">","<","<=",">=". And also it will check, if the entered username and password consists of any of these invalid tokens or malicious keywords by comparing each word with the array of invalid tokens. If they get matched, then user will be considered as malicious and will not be allowed to login into the account.

ISSN: 2277-9655
Impact Factor: 4.116

Fig. 4 Attack is prevented, on applying preventive measures

Case 3: Illegal or Logically incorrect queries

This attack lets an attacker gather important information about the type and structure of the back-end database of Web application. The vulnerability which is exploited by this type of attack is that the default error page returned by application servers is often itself very descriptive i.e. in the form of directory structure as a whole which contains confidential database name and the table names inside it. Additional error information, originally intended to help programmers debug their applications, further helps attackers gain information about the schema of the back-end database. With the intent to perform this attack, an attacker tries to inject statements that cause a syntax, type conversion, or logical error into the database. Syntax errors can be used to identify injectable parameters. Type errors can be used to deduce the data types of certain columns or to extract data. Logical errors often reveal the names of the tables and columns that caused the error. As shown in Fig. 5 below:

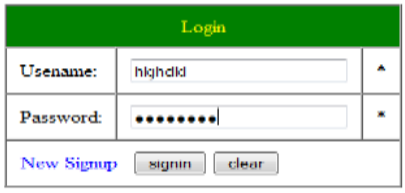

Fig. 5 Illegal query attack by causing syntax error

The error shows the "database name" or the "table name" which is being used in back end database server such as here, "users" is used as table name in the Fig. 6 shown below, which allows an attacker to get access to vulnerable parameters such as table names, column names and its type etc. The attacker can then inject a query to select all rows from that table and can read all confidential information or also an attacker can inject command *"DROP table <tablename> --"* to delete that table permanently from the database which is even more harmful and can cause deletion of all data related to that table. Further attacks can also be possible using table names such as "Union query" attack, which is discussed in next case type.

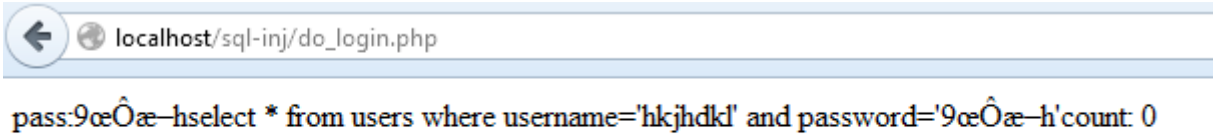

Fig. 6 Syntax error attack revealed table name and field names

Case 4: Union Query

The purpose of this attack is to retrieve confidential information from the tables which we know those are existing at backend, along with the general safe query passed to create an illusion that only single query is used to retrieve non-confidential information. This way, an attacker can trick the database users by passing illegitimate query along with the legitimate one. The screenshot shows the syntax of union query, which will retrieve 'id' from "users" table whose

username is “sonakshi”, also retrieve ‘pin numbers’ from “creditcards” table where name of user is “abcd”, joining the results of above two queries with “union” operator will provide both the id and pin numbers. The important point here is to note that, the return type of both the items such as ‘id’ and ‘pin’ must be same, only then union query will work properly.

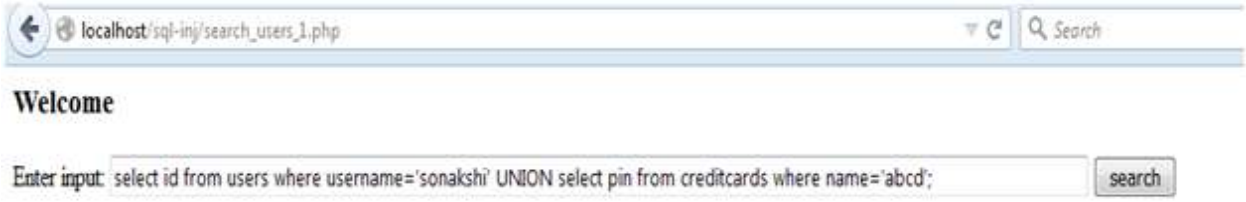

Fig. 7 Union query attack

After entering the union query in the above input field in Fig. 7, and then pressing search button will display the ‘id’ and ‘pin’ returned as a result of union query entered. It is shown in Fig. 8 below:

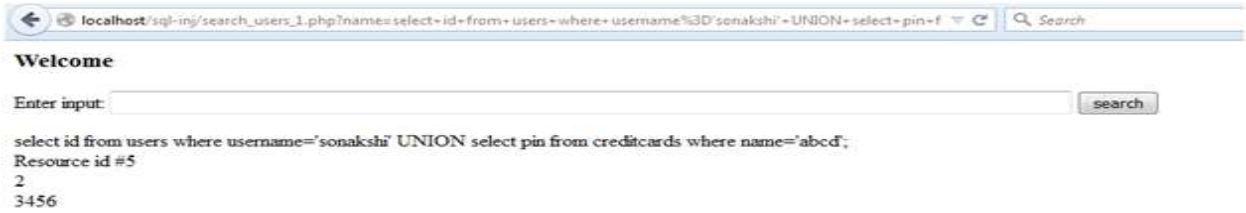

Fig. 8 Result of union query attack

If the name of table and its columns are not known to attacker, then the attacker is not able to do union query attack unless and until he tries to perform this attack by assuming table and column names as hit and trial method.

Prevention:
Union attack can be prevented using a method which detects some common attack keywords such as ‘select’ and ‘union’ keyword especially. By detecting these keywords prior, i.e. when the attacker enters such malicious input in the input or search field with the intent to perform attack, then the user-defined function will automatically detect and prevent such kind of attacks to take place. Also, it can be prevented by making use of parameterized queries.

Case 5: Piggy-Backed queries
In case of piggy-backed query, the attacker tries to append additional query or queries to the original one. The aim of the attacker is to extract the data, adding or modifying the data, performing denial of service and executing remote commands etc. [2] by exploiting the database using query delimiter, such as “;” to append an extra query to the original first query. On successful attack, database receives and executes multiple different queries. In this method of attack, usually the first query is legitimate query, whereas the following queries could be illegitimate. System that is vulnerable to piggy-backed queries is generally due to misconfiguration which allows for multiple statements in one query. The attacker can inject “0; drop table <tablename>--” into pin or password field instead of some logical value. In our framework, a table named “temp” is created as a temporary table used to demonstrate the piggy-backed query attack. The first query is used to fetch all the records of user “sonakshi” whereas, in case of second query “drop” command is used to delete the table “temp” completely, shown in Fig. 9 below.

ISSN: 2277-9655
Impact Factor: 4.116

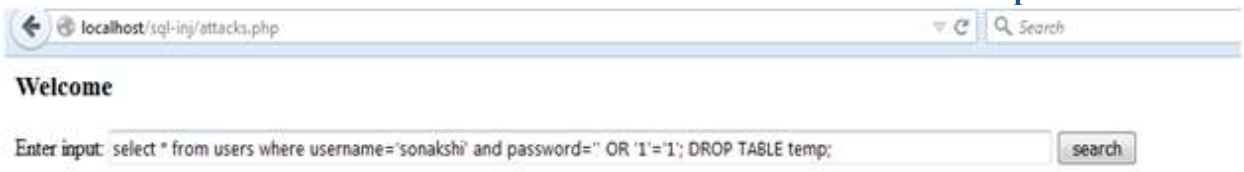

Fig. 9 Demonstration of piggy-backed attack

This attack will not work here because it is already an inbuilt feature of MySQL database as it does not allow to execute multiple queries in one go. If two or more queries are passed in one attempt, then it will throw an error message of mysql_query() function which treats the two queries as one and tries to find out the result but will end up showing no results due to error in accepting more than one query. So, this can be beneficial in case of MySQL database as it will prevent piggy-backed attack in some sense.

Fig. 10 Error while making piggy-backed query attack in MySQL

If single query is used at once in case of MySQL, then it will work. Such as, "DROP TABLE temp;" shown below Fig. 11

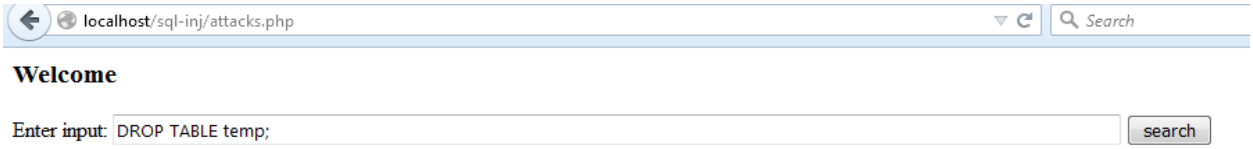

Fig. 11 Single command execution

The result of the above query is successful execution of drop table command when used as running single query at once only. The successful execution is shown in Fig. 12 below where the query result is deletion of table "temp" at backend. This could be a serious harm in case of live websites where deletion of some important and confidential information causes businesses to halt, if cannot be retrieved later. But this could only be possible if the attacker knows the exact name of table being used at backend or if he tries to guess the name.

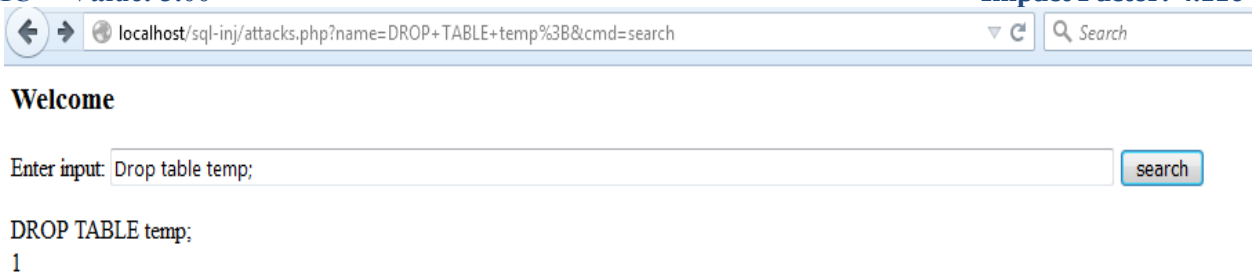

Fig. 12 Table "temp" deleted successfully

Prevention:
Some databases do not require special separation character in multiple queries such as MySQL server allows execution of single query only. So, piggy-backed query attack is not possible in case of MySQL database using PHP. But in case of single DROP table command, we can prevent it by applying a check on certain malicious symbols such as 'drop','select' etc.

Case 6: Attacks done at Password field at login page
SQL injection attacks are also done on the password field to retrieve useful information regarding different users that exist on the database tables. The SQL Injection attack allows external users to read details from the database. In a well designed system this will only include data that is available to the public. In a poorly designed system this may allow external users to discover other users' passwords as well. The attacker can form a malformed query and inject into password field of login interface. The below example shows that the purpose of attacker is to form a Boolean statement which returns all the records of a user "sonakshi", if her password consists of character 'a' somewhere between i.e. %a%. The following example shows the syntax of such attack.
*' OR EXISTS(SELECT * FROM users WHERE name='sonakshi' AND password LIKE '%a%') AND ''='*

Fig. 13 Password injection

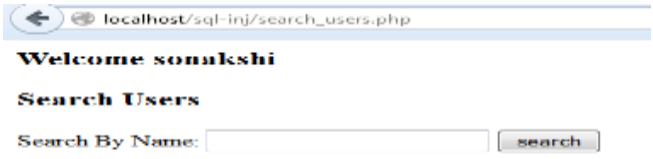

Fig. 14 Attack successful

Prevention:
The injection in password field can be prevented using encryption in password field. If the password is encrypted at the time when user fills the form and submits that, then entered injected password will then be converted into encrypted form and hence its meaning will completely changed to some form other than the Boolean condition. In this way, it does not allow the malicious user to get in as an authenticated one. The Fig. 15 below shows the attacker tries to login by doing injection at password field.

ISSN: 2277-9655
Impact Factor: 4.116

Fig. 15 SQL injection in Password field

But, if encryption is applied to password field, then there will be no injection. Hence, it can be said that encryption can be useful and beneficial in preventing password injections.

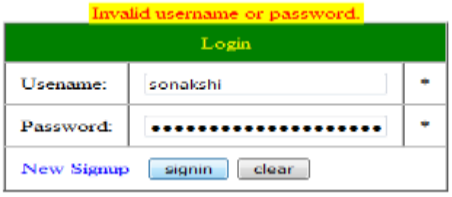

Fig. 16 Encrypted password field

In the above Fig. 16, an error is displayed "Invalid username and password", if password field is encrypted.

SQLIAs ON LIVE WEBSITES

There are some websites which are still vulnerable and hence, the probability of performing attacks on them is very high. Before performing SQLIA on some website or via URL, one must be sure if the site is vulnerable or not. There are some methods to find out, if the site is vulnerable to SQLIA. By making use of those methods, vulnerable sites are found and SQL injection attack is performed using various methods on those sites for the purpose of doing case study. These attacks are shown below.

Fig. 17 Tautology attack on vulnerable website

Fig. 17 shows tautology attack using 'or 1=1 - -' as tautology statement in the username field of a vulnerable website and using any password string because whatever is entered at password field, will be commented out and hence, the attack will be performed and an unauthorized is user is allowed to log in into the admin account without proper authentication. This is because the attack statement caused the query structure to be formed at backend server as in the example stated below.

Select * from user where username = '$username' and password= '$password';

The entered username and password will be embedded into the above query at variable fields. Such as,

*Select * from username = ''or 1=1 - - ' and password= 'anypassword';*

This way the statements written after "- -" will be commented out and only the truth statement i.e. 1=1 will be executed which returns all the records from user table. The meaning of the above final query will be,

*'Select * from users where username= ''or 1=1 - - ';*

It will fetch the complete records of admin account and malicious user can do anything with that.

Fig. 18 User is logged in as admin by unauthorized means

Fig. 19 Viewing Latest news listing at vulnerable site

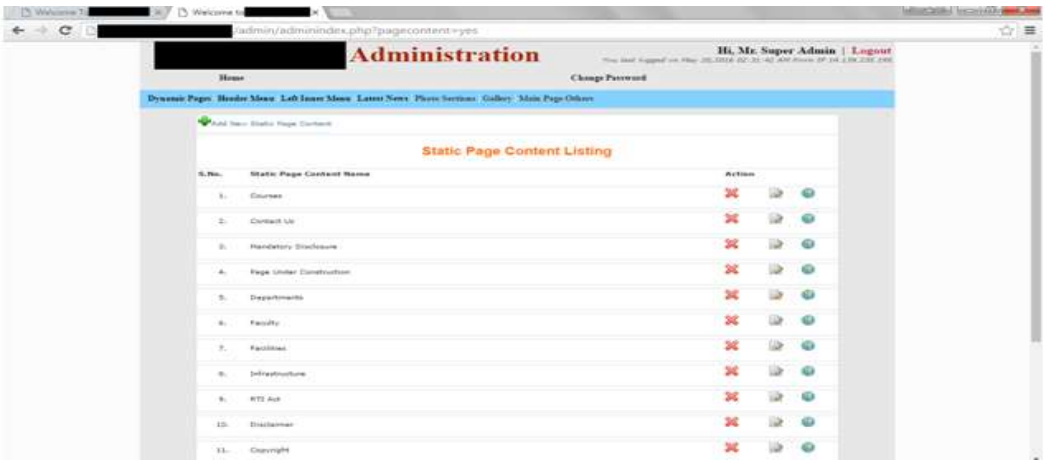

Fig. 20 Static Page content listing

So, here only, the editable content is shown, but not modified it due to security concerns. It is only a demonstration what an attacker can do by impersonating as admin. An attacker can do anything and everything by exploiting the account of admin by various harmful means. The effects of changing it could be very dangerous.

ISSN: 2277-9655
Impact Factor: 4.116

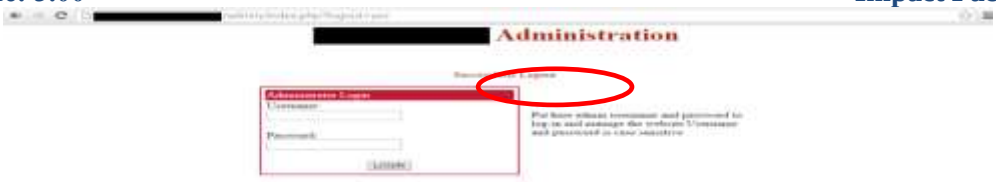

Fig. 21 Attacker successfully logged out of admin account

Another website shown in Fig. 22 which is vulnerable to SQLI attacks is demonstrated below with the help of screenshot, its URL is: www.southbayballet.org/photo-gallery.php?id=38 . Now, the foremost task is to find out whether the given site is vulnerable or not. To do this, a single quote (‘) is appended at the end of URL. If an error message is displayed at the screen, such as *“You have an error your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '" at line 1”,* then the given website is vulnerable. Also here, it shows the additional information about the table name such as southbay_showimages and column names such as image_id, show_id. We can use this information for some illegal means such as tables can be deleted from the database.

Fig. 22 Vulnerable website

Upon appending single quote (‘) at the end of URL, we will be able to identify whether the particular web application is vulnerable to SQLI attack or not. The following screenshots will show the step by step procedure to do attack via URL of a vulnerable website.

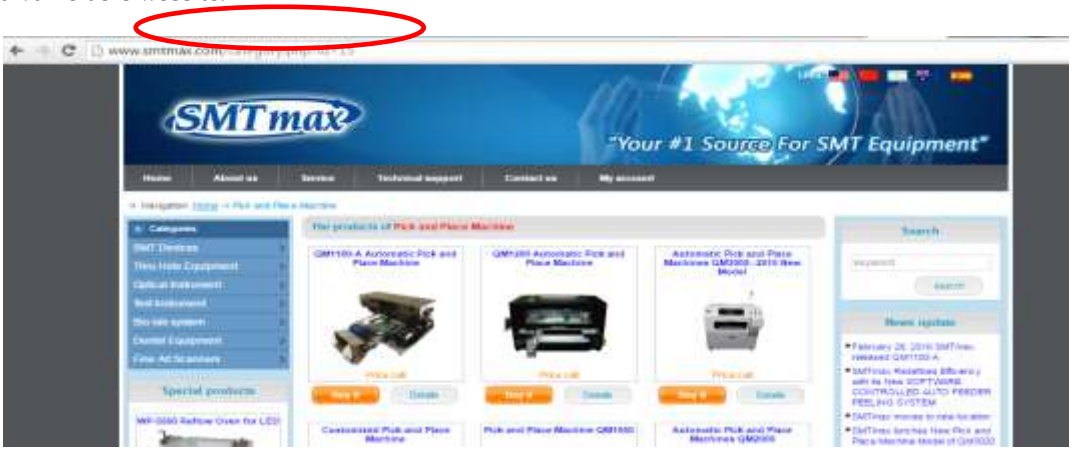

Fig. 23 SQLI Vulnerable website

Here's the URL of a website which is vulnerable to attack: http://www.smtmax.com/category.php?id=15, which is highlighted in above Fig. 23. Following steps are to be followed to perform URL SQL injection.

i) Append single quote (') at the end of URL, if it is vulnerable to attack, then a message will be displayed on screen e.g. "You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '" at line 1". This error means this site is vulnerable and we can perform further injections on it.

ii) In order to find out number of columns in a particular table in the current database, "order by" command is used. E.g. <complete URL><space><order by n- -> i.e. keep trying some particular number at the place of n. Start from 1 and go on till an error of unknown column is displayed. It means, total there are "n-1" no. of columns in a particular table.

iii) Union command is used such as <complete URL with negative id value>and<false condition><union select 1,2,3,4,5,…,n-1- -> E.g. www.smtmax.com/category.php?id= **-n and 1=2 union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14- - .** Some number will be displayed on screen which will be used in further steps. Let say "3".

iv) Now, version of the database can be find out by writing **version()** in place of that number i.e."3" in the URL and the remaining URL is typed same as above. In our example, database version will be displayed on screen such as **'5.6.29-log'**.

v) Similarly, name of the database can be find out by writing **database()** in place of version() in the above step. E.g. **omxiec5_smtmax**. Also which client is in use will be easily find out by typing **user()** in place of database() in above step. E.g. **omxiec5_smtmax@localhost.**

The following figures will describe the above steps.

Fig. 28 Database is shown using database() function injection in URL

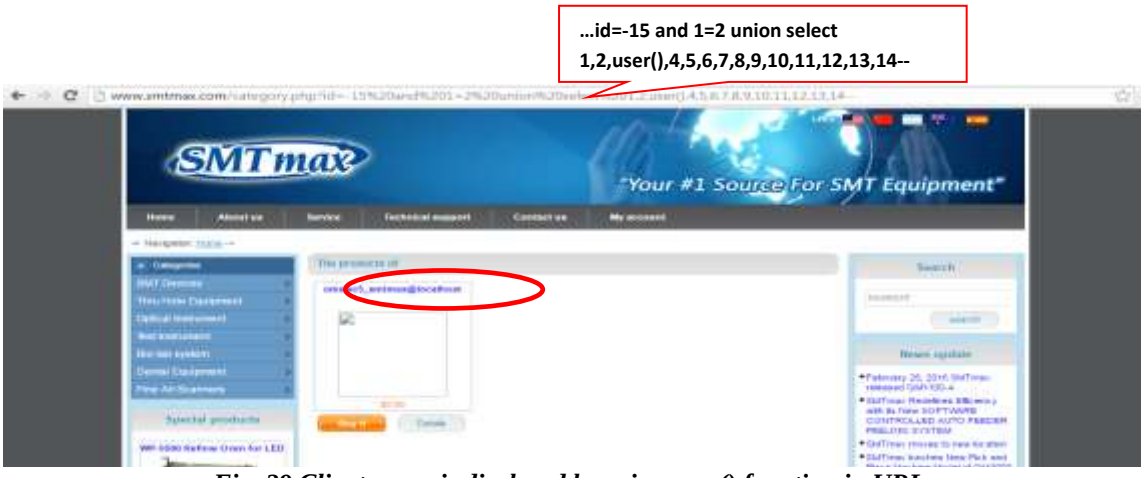

Fig. 29 Client name is displayed by using user() function in URL

GUIDELINES TO DETECT, PREVENT AND MITIGATE SQL INJECTION ATTACKS

SQL injection attacks have become more common, more ambitious and increasingly sophisticated. So, there is need to find an effective and feasible solution for this problem in web applications security. Following section will describe certain set of guidelines for how to detect, prevent and mitigate SQLIAs in web applications.

Guidelines to detect & prevent SQLIAs before occurring are as follows:

i) ***Detection of malicious symbols:*** Certain checks should be applied to detect some known malicious symbols or keywords such as "'", "or", "select", "=", "union", "drop" etc. These are responsible for an attack to take place.

ii) ***Prevention of attacks performed via URLs:*** To prevent this type of attack, the programmer should always make use of HTTP POST method instead of HTTP GET method because information sent from a form with the GET method is **visible to everyone**. This is the point which could be easily exploited by attacker to inject malicious SQL queries and extract sensitive database information.

iii) ***Keep the page names other than admin:*** One can prevent lot of attacks through SQLIs by keeping the name of admin page something else other than "admin". The attacker will first try to locate admin page of a website, by entering the complete URL and then appending "/admin". It is first and foremost mistake done by administrator to keep the admin handled login page name same as "admin". So, its name should be something else which cannot be easily speculated by attacker.

iv) ***Escaping all user supplied input:*** This technique is used to escape user input before putting it in a query. Each DBMS supports one or more character escaping schemes specific to certain kinds of queries. It is then used to escape all user supplied input using the proper escaping scheme for the particular kind of database.

Guidelines to be followed to prevent & mitigate SQLIAs at login forms are given below:

i) ***Making use of Selects, radio buttons, and checkboxes:*** Inputs from user via input fields such as text fields and password fields etc. could be more harmful and is vulnerable to attack, than inputs through check boxes, radio buttons and drop down lists etc. So, developer should create such functionality in user registration form and in login form to accept input from user via radio buttons, check boxes or drop down menus.

ii) ***Accept known good by incorporating input validation:*** Validation is often carried out in two ways: by blacklisting dangerous or unwanted characters and by white listing only those characters that are allowed in a given circumstance. The checking should be done that the data is one of a set of tightly constrained known good values. Any data that doesn't match should be rejected. Data should be:

- Strongly typed at all times
- Length checked and fields length minimized
- Range checked if a numeric
- Unsigned unless required to be signed
- Syntax or grammar should be checked prior to first use or inspection
- Integrity checks

iii) ***Sanitize the input data:*** Sanitization usually involves running any submitted data through a function such as MySQL's mysql_real_escape_string () function to ensure that any dangerous characters (like " ' ") are not passed to a SQL query in data.

iv) ***Allowing single words input only:*** If input from users is allowed then it must not be separated with spaces, single quotes, use of logical characters such as "or" & "AND" etc. As, allowing the use of such characters, will increase the possibilities of attack through input fields of login form.

v) ***Prevent Parameter Tampering:*** This includes intentionally tampering resistant fields such as checkboxes, radio buttons, drop down lists etc. by rewriting any client side HTML to suit the attacker.

vi) ***Use Parameterized Queries:*** MySQL, like many database systems, supports a concept called parameterized queries. This is where the SQL query uses a parameter instead of injecting the values directly into the query. This means defining the SQL code that is to be executed with placeholders for parameter values, then user input is passed as parameters along with its data type defined.

vii) ***Use of SQL Stored Procedures:*** Stored procedures seem to be a wonderful panacea against injection attacks. It adds an extra layer of abstraction in to the design of a software system. This means, as long as the interface on the stored procedure remains the same, the underlying table structure can change with no noticeable consequences to the application that is using the database.

Guidelines to be followed to prevent SQLIA after it had occurred are given below:

Encrypting user credentials

If by any chance attacker has tried to gain unauthorized access to some authenticated user's account and also access to database and its contents, then the data should be present in encrypted form at the backend to prevent an attacker to read confidential information from database tables. Using encryption, the encrypted values of username and password along with original username are inserted into user table. By some means, if attacker tries to get access to database and its table, then encrypting data values will prevent attacker to read such sensitive data and hence, any further damage such as modifying the table contents such as columns and its values would have no effect.

CONCLUSION

Web sites are static, dynamic and most of the time a combination of both. Web applications need protection in their database to ensure security. SQL injection attacks allow attackers to spoof identity, tamper with existing data, can cause repudiation issues such as voiding transactions or even changing balances, allow the complete disclosure of all data, destroy the data or make it otherwise unavailable, and become administrators of the database server. If an application fails to properly construct SQL statements it is possible for an attacker to alter the statement structure and execute unplanned and potentially hostile commands. It has become a common issue with database-driven web sites as it can be detected and easily exploited, and as such, any site or software package with even a minimal user interaction is likely to be subject to an attempted attack of this kind**.** In this paper, a case study is provided with clear distinction among all types of SQL injections these days. Also, attacks on various live websites will provide an understanding to programmer that what are the vulnerabilities which are still prevailing in the web applications. Finally, a set of guidelines has been devised in section 5, to prevent, detect and mitigate the effect of SQL injection attacks. The aim is to prevent SQLIAs from occurring and also to make it less severe for future applications access by users.

REFERENCES

[1] Ava Fedorov. (2014) SC Magazine UK. [Online]. http://www.scmagazineuk.com/sql-injection-attacks-still-a-major-threat/article/347374/

[2] V. Nithya, R. Regan, and J. Vijayaraghavan, "A survey of SQL injection attacks, their Detection and Prevention techniques," International Journal of Engineering and Computer Science(IJECS), vol. 2, no. 4, April 2013.

[3] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso, "A Classification of SQL Injection Attacks and Countermeasures," IEEE, 2006.

[4] Sruthy Manmadhan and T. Manesh, "A Method of detecting SQL Injection attack to secure web applications," International journal of Distributed and Parallel Systems(IJDPS), vol. 3, no. 6, November 2012.

[5] Aanal Bhanderi and Nency Rawal, "A review and Detection mechanism for SQL injection attacks," International Journal of Innovative Research in Science, Engineering & Technology, vol. 4, no. 12, pp. 12446-12452, December 2015.

[6] Pupendra Kumar and R.K. Pateriya, "A Survey on SQL injection attacks, detection and prevention techniques," IEEE ICCCNT, 2012.

[7] Atefeh Tajpour, Suhaimi Ibrahim, and Mohammad Sharifi, "Web Applications Security by SQL Injection Detection Tools," International Journal of Computer Science Issues(IJCSI), vol. 9, no. 2, pp. 332-339, March 2012.

[8] Mahima Srivastava, "Algorithm to Prevent backend database against SQL injection attacks," IEEE, pp. 754-757, 2014.

[9] Pankajdeep Kaur and Kanwalpreet Kaur, "SQL Injection: Study and Augmentation," International Conference on Signal Processing, Computing and Control (2015 ISPCC), 2015.

[10] Sonakshi, Rakesh Kumar, and Girdhar Gopal, "Pprevention of SQL Injection Attacks using RC4 and Blowfish Encryption Techniques," International Journal of Enineering Research & Technology (IJERT), vol. 5, no. 06, pp. 25-29, June 2016.
[11] Shubham Mukherjee, Sudeshna Bora, Pritam Sen, and Chittaranjan Pradhan, "SQL Injection: A Sample Review," 6th ICCCNT, Denton, U.S.A., December 2015.
[12] Chandrashekhar Sharma and Dr. S.C. Jain, "Analysis and Classification of SQL injection vulnerabilities and Attacks on Web Applications," IEEE International Conference on Advances in Engineering and Technology Research(ICAETR 2014), 2014.
[13] Debrata Kar and Suvasini Panigrahi, "Prevention of SQL Injection attack using query transformation and hashing," 3rd IEEE International Advance Computing Conference (IACC), pp. 1317-1323, 2013.