ISSN: 2321-7782 (Online)
Volume 3, Issue 5, May 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

A New Proposed Method for Detection and Prevention of SQLIA (SQL Injection Attack)

Parveen Sadotra
(CEH)
Teaching Assistant & (CEH), GDC,
R.S. Pura, Jammu (J&K), India

Abstract: ***SQL injection attacks pose a serious security threat to Web applications. This type of attack allows attackers to obtain unauthorized access to the databases underlying the applications and to the potentially sensitive information these databases contain. Although researchers and Security experts have proposed various methods to address the SQL injection problem, current approaches either fail to address the full scope of the problem or have limitations that prevent their use and adoption. Many researchers and experts are familiar with only a subset of the wide range of techniques available to hackers who are try to take advantage of SQL injection vulnerabilities. As a consequence, many solutions proposed in the literature address only some of the issues related to SQL injection. To address this problem, we present an extensive review of the different types of SQL injection attacks known to date. And propose a new method detection d and prevention of SQLIA.***

Keywords: Web application, SQLIA, detection, prevention, vulnerabilities

I. INTRODUCTION

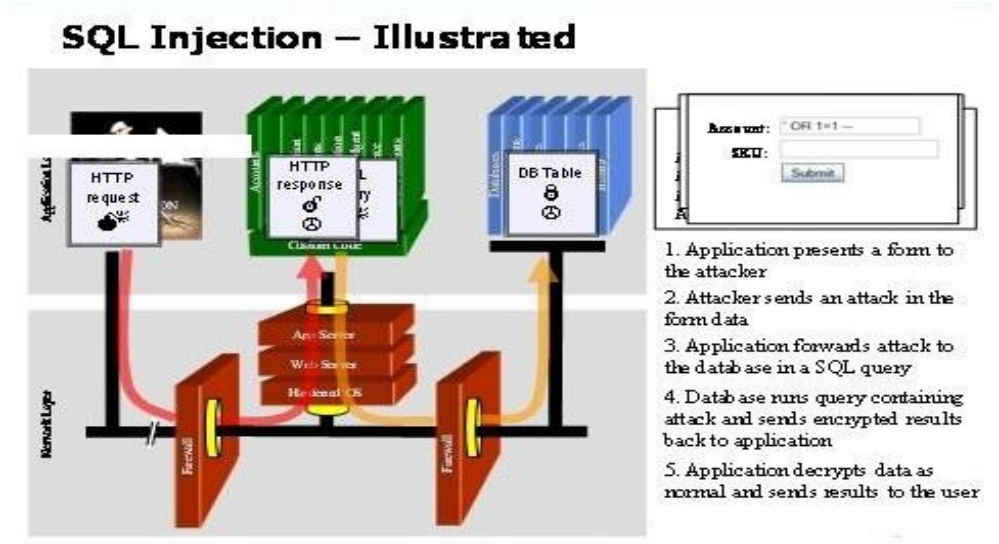
Nowadays, for most of our activities in our life, we are depending on internet or website/web applications. There is a natural trend in the human world that as the usage of a particular service increases; the attacker's interest in it also increases. Exactly same thing happened in case of web applications. There aremany types of attacks against web applications which attackers user but, SQL Injection Attack (SQLIA) is one of the top most threats against these. So in the in the present situation, it is highly required to have a good solution to prevent such types of attack to secure the information in our database.

SQL Injection targets the web applications that use a back end database. Working of a typical web application is as follows:

» User is giving request through web browsers, which may be some parameters like username, password, account number etc. These are then passed to the web application program where some dynamic SQL queries are generated to retrieve required data from the back end database.

» SQL Injection attack is launched through especially designed user inputs. That is attackers are allowed to give requests as normal users. Then they intentionally create some bad input patterns which are passed to the web application code. If the application is vulnerable to SQLIA, then this specially created input will change the intended structure of the SQL query that is being executed on the back end database and will affect the security of information stored in the database. The tendency to change the querystructure is the most characteristics feature of SQLIA which is being used for its prevention also.

For better understanding let us have a look at the following illustration.



II. Need / Importance Of The Study

SQL Injection is one of the most common application layer attacks currently being used on the websites. Despite the fact that it is relatively easy to protect against SQL Injection, there are a large number of web sites/applications that remain vulnerable. According to the Web Application Security Consortium (WASC) 10% of the total hacking incidents reported were due to SQL Injection. More recent data from our own research shows that about 50% of the websites we have scanned this year are susceptible to SQL Injection attack.

It may be difficult to answer the question whether your web site and web applications are vulnerable to SQL Injection or not especially if you are not a programmer or you are not the person who has coded your web applications by your own but our experience leads us to believe that there is a significant chance that your data is already at risk from SQL Injection. Whether an attacker is able to see the data stored on the database or not, in fact it depends on how your website is coded to display the results of the queries sent by user. What is certain is that the attacker will be able to execute arbitrary SQL Commands on the vulnerable system, either to compromise it or else to obtain information.If website is improperly coded then you have the risk of having your customer and company data compromised. Even if an attacker is not able to modify the system, he would still be able to read valuable information contained in it.What an attacker gains access to, it also depends on the level of security set by the database. The database could be set to restrict to certain commands only. A read access normally is enabled for use by web application back ends for better security.

III. Statement Of The Problem

An SQL injection attack involves the alteration of SQL statements that are used within a web application through the use of attacker-supplied data. Insufficient input validation and improper construction of SQL statements in web applications can expose them to SQL injection attacks. SQL injection is such a prevalent and potentially destructive attack that the Open Web Application Security Project lists it as the number one threat to web applications.

Ramifications of Successful SQL Injection Attacks

Although the effects of a successful SQL injection attack vary based on the targeted application and how that application processes user-supplied data, SQL injection can generally be used to perform the following types of attacks:

» **Authentication Bypass:** This attack allows an attacker to log on to an application, potentially with administrative privileges, without supplying a valid username and password.

- **Information Disclosure:** This attack allows an attacker to obtain, either directly or indirectly, sensitive information in a database.
- **Compromised Data Integrity:** This attack involves the alteration of the contents of a database. An attacker could use this attack to deface a web page or more likely to insert malicious content into otherwise innocuous web pages. This technique has been demonstrated via the attacks that are described in Mass exploits with SQL Injection at the SANS Internet Storm Center.
- **Compromised Availability of Data:** This attack allows an attacker to delete information with the intent to cause harm or delete log or audit information in a database.
- **Remote Command Execution:** Performing command execution through a database can allow an attacker to compromise the host operating system. These attacks often leverage an existing, predefined stored procedure for host operating system command execution. The most recognized variety of this attack uses the xp_cmdshell stored procedure that is common to Microsoft SQL Server installations or leverages the ability to create an external procedure call on Oracle databases.

An Example of SQL Injection for Authentication Bypass

One of the many possible uses for SQL injection involves bypassing an application login process. The following example illustrates the general operation of a SQL injection attack. The following HTML form solicits login information from an application user. Although this example uses an HTTP POST request, an attacker could also use HTML forms that use the HTTP GET method.

<form action="/cgi-bin/login" method=post>

Username: <input type=text name=username>

Password: <input type=password name=password>

<input type=submit value=Login>

When a user enters his or her information into this form and clicks **Login**, the browser submits a string to the web server that contains the user's credentials. This string appears in the body of the HTTP or HTTPS POST request as:

username=*submittedUser*&password=*submittedPassword*

An application with a vulnerable login process may accept the submitted information and use it as part of the following SQL statement, which locates a user profile that contains the submitted username and password:

select * from Users where (username = '*submittedUser*' and password = '*submittedPassword*');

Unless an application uses strict input validation, it may be vulnerable to a SQL injection attack. For example, if an application accepts and processes user-supplied data without any validation, an attacker could submit a maliciously crafted username and password. Consider the following string sent by an attacker:

username=admin%27%29+--+&password=+

Once this string is received and URL-decoded, the application will attempt to build a SQL statement using a username of *admin')* -- and a password that consists of a single space. Placing these items into the previous SQL statement yields:

select * from Users where (username = '*admin*') -- and password = ' ');

As the previous example demonstrates, the attacker-crafted username changes the logic of the SQL statement to effectively remove the password check. In the above example, an attacker could successfully log in to the application using the admin account without knowledge of the password to that account.

The string of two dash characters (--) that appears in the crafted input is very important; it indicates to the database server that the remaining characters in the SQL statement are a comment and should be ignored. This capability is one of the most important tools that are available to an attacker and without it; it would be difficult to ensure that the malicious SQL statements were syntactically correct.

Although the crafted field, which is the username field in the previous example, must be tailored to the vulnerable application, a large set of documented strings that are readily available on the Internet have proven successful at enabling SQL injection attacks. The previous example may be simplistic, but it illustrates the effectiveness of SQL injection attack techniques.

IV. OBJECTIVES

It has been over a decade since the original research paper on SQL Injection was published. Over the years the SQL Injection threat has grown to the point where now we are seeing weaponized SQL Injection attacks perpetrated by state actors. Organizations are continually being breached via SQL Injection attacks that slip seamlessly through the firewall over port 80 for HTTP or 443 for SSL to soft internal networks and vulnerable databases. Objective of this research paper to see and analyze threat of SQL injection in our web applications. There are many methods of preventing and avoiding this attack but here we will see latest and most effective method to counter this SQLIA.

V. HYPOTHESIS

The basic type of SQL injection consists of direct insertion of code into user input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata,when the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed.

The injection process works by prematurely terminating a text string and appending a new command. Because the inserted command may have additional strings appended to it before it is executed, the malefactor terminates the injected string with a comment mark "--". Subsequent text is ignored at execution time

Types SQLIA

There are many types of SQL Injection attacks and attack depends on the intent of attacker. The attacker can perform the attack sequentially or altogether. We can classify the SQLIA into following types: -

1. **Tautologies** The general goal of a tautology-based attack is to injectcode in one or more conditional statements so that they always evaluate to true. The consequences of this attack depend on how the results of the query are used within the application. The most common usages are to bypass authentication pages and extract data. In this type of injection, an attacker exploits an injectable field that is used in a query's WHERE conditional. Transforming the conditional into a tautology causes all of the rows in the database table targeted by the query to be returned.

2. **Union Query** In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>.

3. **Illegal/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 makes a type mismatch error by injecting the following text into the pin input field.

4. **Piggybacked Queries** In this attack type, an attacker tries to inject additionalqueries into the original query. We distinguish this type from othersbecause, in this case, attackers are not trying to modify the originalintended query; instead, they are trying to include new and distinctqueries that piggy-back.On the original query. As a result, thedatabase receives multiple SQL queries. The first is the intendedquery which is executed as normal; the subsequent ones are theinjected queries, which are executed in addition to the first.

5. **Inference in this attack** The query is modified to recast it in the form of an action that is executed based on the answer to a true/false question about data values in the database. In this type of injection, attackers are generally trying to attack a site that has been secured enough so that, when an injection has succeeded, there is no usable feedback via database error messages. Since databaseerror messages are unavailable to provide the attacker with feedback, attackers must use a different method of obtaining a response from the database. In this situation, the attacker injects commandsinto the site and then observes how the function/response of the website changes.

6. Stored procedure is a part of database that programmer could set an extra abstraction layer on the database. As stored procedure could be coded by programmer, so, this part is as inject able as web application forms. Depending on specific stored procedure on the database there are different ways to attack.

VI. Research Methodology

For our research work, we contacted web developers and security professionals and asked them about various issues faced in preventing SQL injection attack. We found various methods of SQL injection attacks and analyzed them. After closely investigating all the types of attacks we could conclude the effect of these attacks then we went through all the security measures taken by various web developers and professionals. At last we proposed a new method for detection and prevention of SQL injection attacks.

VII. Results, Findings And Discussion

Proposed techniques of SQLIA were compared to assess whether it was capable of addressing the different attack types mentioned above. It is noticeable that this comparison is based on the articles not empirically experience. Here is the result of our research: -

Sl No	Type of Attack	Techniques that can stop all attacks	Techniques that address the attack type partially	Techniques that could not stop attacks	Remarks
1	Taut	60 %	35 %	5 %	
2	Illegal/Incorrect	55 %	35 %	10 %	
3	Piggy Back	60 %	35 %	5 %	
4	Union	60 %	35 %	5 %	
5	Stored Proc	25 %	35 %	40 %	
6	Infer	60 %	35 %	5 %	
7	Alter Encoding	50 %	35 %	15 %	

The results of our study are very encouraging. For all subjects, our technique was able to correctly identify all attacks as SQLIAs, while allowing all legitimate queries to be performed. In other words, for the cases considered, our technique generated nofalse positives and no false negatives. The lack of false positives and false negatives is promising and provides evidence of the viability of the technique.

PROPOSED MODEL

After studying the work of various researchers, we propose the methods by which the SQLIA can be prevented easily, for this purpose we propose some defense mechanism and username and password validation using cryptographically Hash function.

a) SQLIA Prevention using Data Validation –

For validation of data we propose the following three approaches:

a) escape single quotes

```
functionescape ( input )
input = replace(input, —’||, —’’||)
escape = input
end function
```

b) Reject input that is known to be bad

```
functionvalidate string( input )
known_bad = array( "select", "insert", "update", "delete", "drop", "--", "'" )
validate_string = true
for i = lbound( known_bad ) to ubound( known_bad )
if ( instr( 1, input, known_bad(i), vbtextcompare ) !=0 ) then
validate_string = false
exit function
end if
next
end function
```

c) Accept only input that is known to be good –

```
functionvalidatepassword( input )
good_password_ch="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
validatepassword = true
for i = 1 to len( input )
c = mid( input, i, 1 )
if ( InStr( good_password_ch, c ) = 0 ) then
validatepassword = false
exit function
end if
next
end function
```

b) SQLIA Prevention using Crypto graphical Hash function –

For the purpose of protecting against unauthorized login by using SQLIA, we provide Hash function mechanism. In this mechanism we need to add two additional attributes in the login database one for Hash value for user name and another is for has value for password. When the admin first creates the user account and assigns a password the Hash value is automatically generated by using Hash function algorithm and stored in the database along with the login information of the user. This information is stored in the database in encrypted form. Now when the user needs to login to the server he/she passes his/her

username and password and the Hash value is generated automatically and the Hash value is also sent from client computer to server along with the username and password in encrypted form as HTTP request.

VIII. RECOMMENDATIONS / SUGGESTIONS

The most important precautions are data sanitization and validation, which should already be in place. Sanitization usually involves running any submitted data through a function to ensure that any dangerous characters (like " ' ") are not passed to a SQL query in data. Following are the suggestions to prevent SQL injection attacks on our websites/ web applications: -

- **Trust no-one:** Never trust on everyone, assume all user submitted data is evil and validate and sanitize everything.
- **Don't use dynamic SQL if it can be avoided:** use prepared statements, parameterized queries or stored procedures instead whenever possible.
- **Update and patch**: vulnerabilities in applications and databases that hackers can exploit using SQL injection are regularly discovered, so it is recommended to apply patches and updates as soon as possible and practical.
- **Web Application Firewall:** You must consider a web application firewall, either software or appliance based to help filter out malicious data. Good ones will have a comprehensive set of default rule and make it easy to add new ones whenever necessary.
- **Reduce your attack surface:** You should get rid of any database functionality that you don't need to prevent a hacker taking advantage of it. For example, the xp_cmdshell extended stored procedure in MS SQL spawns a Windows command shell and passes in a string for execution, which could be very useful indeed for a hacker. The Windows process spawned by **xp_cmdshell** has the same security privileges as the SQL Server service account.
- **Use appropriate privileges:** don't connect to your database using an account with admin level privileges unless there is some compelling reason to do so. Using a limited access account is always safer and it can limit what a hacker is able to do.
- **Secrets means secret:** Keep your secrets, secrets, assume that your application is not secure and act accordingly by encrypting or hashing passwords and other confidential data including connection strings.
- **Don't divulge more information than you need to:** Attackers can learn a great deal about database architecture from error messages so ensure that they display minimal information. Use the "RemoteOnly" customErrors mode to display verbose error messages on the local machine while ensuring that an external hacker gets nothing more than the fact that his actions resulted in an unhandled error.
- **Never forget the basics:** You should Change the passwords of application accounts into the database regularly. This is common sense, but in practice these passwords often stay unchanged for a very long time.
- **Buy better software:** Make code writers responsible for checking the code and for fixing security flaws in custom applications before the software is delivered.

IX. CONCLUSION

SQL Injection poses a serious security issue over the Internet or over web application. In SQL injection attacks, hackers can take advantage of poorly coded Web application software to introduce malicious code into the organization's systems and network. The vulnerability exists when a Web application do not properly filter or validate the entered data by a user on a Web page. Large Web applications have hundreds of places where users can input data, each of which can provide a SQL injection opportunity. Attacker can steal confidential data of the organization with these attacks resulting loss of market value of the organization. This paper presents an effective survey of SQL Injection attack, detection and prevention techniques.

X. SCOPE FOR FURTHER RESEARCH

Here we have developed a highly automated approach for protecting Web applications from SQLinjection attacks. This application consists of

» Using regular expression we found known attacks

» Allowing only trusted data to form the semantically relevant parts of queries such as SQL keywordsand operators.

» Performs syntax-aware evaluation of a query string immediately before the stringis sent to the database for execution. This paper also provides practical advantages over the manyexisting techniques whose application requires customized and complex runtime environments

To implement the functionality of query fragments that come from other sources, web application developer mustlist these sources in a configuration file that SQLIMDS processes before instrument the application.We can enhance SQLIMDS to work on web applications developed using any programminglanguage or framework. The work can be extended by using SQLIMDS to protect actually deployedWeb applications, Implementation for binary applications. for high availability on load balancing anddisaster recovery. Load balancing can automatically handle failures. (bad disks, failing fans, "oops,unplugged the wrong box", …..), Make this service always work any IP addresses, Anything that hasfail over or an alternate server the IP needs to move (much faster than changing DNS).In DisasterRecovery Planning can have a status update site / weblog, Plans for getting hardware replacements,Plans for getting running temporarily on rented "dedicated servers".

References

1. <http://link.springer.com/chapter/10.1007%2F978-3-642-22786-8_13>
2. http://airccse.org/journal/ijdps/papers/3612ijdps01.pdf
3. Parveen Sadotra (CEH), Dr. Anup Girdhar, 2013, "Information Technology (Amended) Act 2008: A Critical Analysis", 'Cyber Times International Journal of Technology & Management', Vol 6. Issue 2. Pg.108-112.
4. <http://www.cc.gatech.edu/~orso/papers/halfond.viegas.orso.ISSSE06.pdf>
5. <http://www.aicit.org/ijact/ppl/ijact_binder_august_11.pdf>
6. <http://www.ijcce.org/papers/244-E091.pdf>
7. Parveen Sadotra (CEH), Dr. Anup Girdhar, 2013, "Ontology Based Intrusion Detection systems", 'Cyber Times International Journal of Technology & Management', Vol 6. Issue 2. Pg.141-147.
8. <http://www-bcf.usc.edu/~halfond/papers/halfond07springer.pdf>
9. <http://www.sciencedirect.com/science/article/pii/S1877705812008600>
10. <http://www.computerweekly.com/tip/SQL-injection-detection-tools-and-prevention-strategies>
11. Parveen Sadotra (CEH), Dr. Anup Girdhar, 2013, "Penetration Testing/Cyber security Assessment – XYZ Company", 'Cyber Times International Journal of Technology & Management', Vol 6. Issue 1. Pg.340-350.
12. <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6396096&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6396096>
13. <http://www.dbnetworks.com/pdf/sql-injection-detection-web-environment.pdf>
14. <https://www.acunetix.com/websitesecurity/sql-injection/>
15. <http://www.cisco.com/web/about/security/intelligence/sql_injection.html>
16. <http://www.enterprisenetworkingplanet.com/netsecur/article.php/3866756/10-Ways-to-Prevent-or-Mitigate-SQL-Injection-Attacks.htm>
17. <http://searchsecurity.techtarget.com/tutorial/SQL-injection-protection-A-guide-on-how-to-prevent-and-stop-attacks>

