International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-3, September 2019

SQL Injection Detection and Prevention Techniques in ASP.NET Web Application

Mohammad Abu Kausar, Mohammad Nasar, Aiman Moyaid

Abstract: ***Injection in SQL (structure query language) is one of the threats to web-based apps, mobile apps and even desktop applications associated to the database. An effective SQL Injection Attacks (SQLIA) could have severe implications for the victimized organization including economic loss, loss of reputation, enforcement and infringement of regulations. Systems which do not validate the input of the user correctly make them susceptible to SQL injection. SQLIA happens once an attacker can incorporate a sequence of harmful SQL commands into a request by changing back-end database through user information. To use this sort of attacks may readily hack applications and grab the private information by the attacker. In this article we introduce deferential sort of process to safeguard against current SQLIA method and instruments that are used in ASP.NET apps to detect or stop these attacks.***

Index Terms: ***SQL Injection, Cybercrime, SQLIA, Vulnerabilities, ASP.NET web application, Web Application Security.***

I. INTRODUCTION

Nowadays internet is growing at very high speed, internet providers are now using web applications to introduce the utilities and use the web framework to become an exciting approach for software applications businesses. It enables the structure of ubiquitous apps that millions of users can use easy. World Wide Web (WWW) has been a wonderful development, but at the same time internet attacks have increased. There will be millions of contraventions of safety happening day after day. According to Akamai [1], however, in the 3rd quarter of 2017, the amount of internet application attacks improved by 69% compared to Q3 2016. Whereas SQL injection and local File Incorporation attacks accounted for 85% of all of these assaults, XSS only accounted for 9%. For the 2017 Data Breach Study of Verizon [2], just 15.4 percent of recorded events involving web application assaults, while 29.5 percent of breaches were suffered by web application attacks. SQL Injection is the most prevalent attack on the internet. Figure 1 demonstrates the recent SQL Injection report.

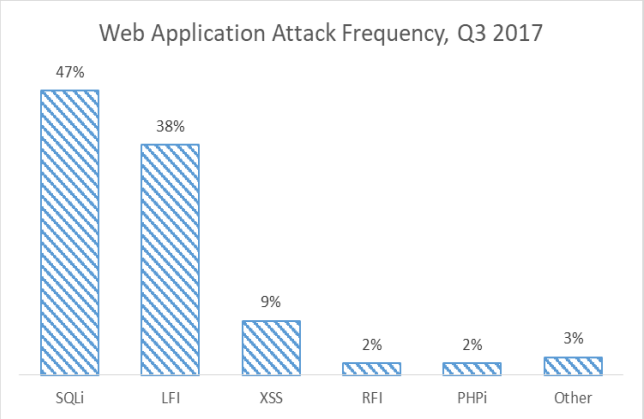

Figure 1: SQL injection and Local File Inclusion attacks accounted for 85% of web application attacks in Q3 2017 [1].

As per the studies conducted by US-based cloud service vendor Akamai, who discovered in his Internet State of Report [13] that SQL injection and Local Filer Inclusion attacks accounted for more than 85 percent of the vectors of assault reported. From November 2017 to March 2019, SQL injection attacks accounted for 65% of web-based attack vectors.

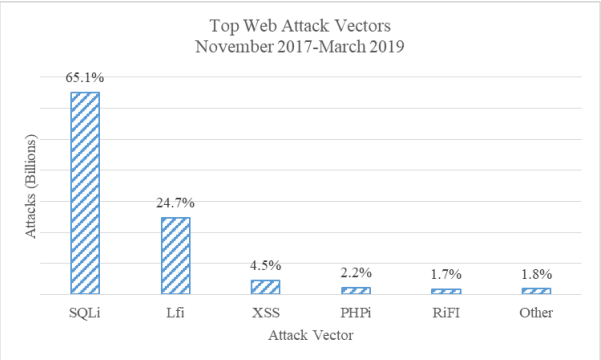

Figure 2: Top Web Attack Vectors from November 2017 – March 2019 [13].

Akamai reported internet attack warnings of just under four billion (3,993) over a 17-month duration, of which only 1,23 billion occurred during the first half of 2019.

Traditional SQL injections seemed to be simple to avoid, identify, and many processes have been addressed, techniques to defeat SQL injections. According to comprehensive studies by Howard [3] and his team, the multiple methodologies used to solve the attack are safe code.

Manuscript published on 30 September 2019

Mohammad Abu Kausar*, Department of Information Systems, University of Nizwa, Nizwa, Sultanate of Oman.

Mohammad Nasar, Department of Computing and Informatics, Mazoon College, Muscat, Sultanate of Oman.

Aiman Moyaid, Department of Information Systems, University of Nizwa, Nizwa, Sultanate of Oman.

This refers to the use of encoding and decoding methods to write defensive code with adequate validation [3].

Moreover, efficient safety mechanisms appear to be very crucial for web applications and resolving them. Mostly all web application is build based on 3 Tier architecture, 1. User Interface 2. Business logic layer and 3. Database layer. Figure 3 illustrate about three tier architecture in any web application. SQL injection is an attack method that exploits a vulnerability that occurs in an application's database layer [4]. One of the targets of SQLIA is the web application along with a backend that stores significant data, as the databases are directly accessible to an attacker by inserting SQL queries that the web application obtains. Due to the frequent retention of user information in these databases, significant information is lost and security breaches.

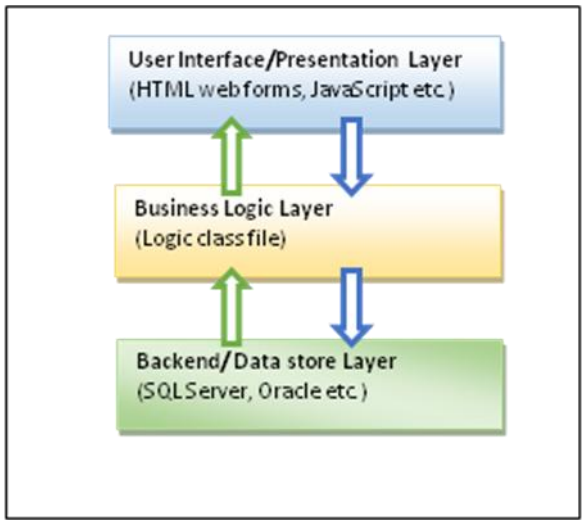

Figure 3: 3 tier architecture of Web Application

By injecting SQL statements that are obtained by the web application, important databases are completely available to the attacker. In case of university or college database, student Information is often stored in such databases; there is a loss of significant data and a violation of safety. Attackers could even use an exposure to SQL injection that attackers use to regulate and make the structure of the internet application very week.

SQL Injection points out a class of code-injection assaults; Student provides the data that is integrated into the SQL query so that portion of the student's data is recognized through SQL query. Through these vulnerabilities, SQL commands provided to the database by the attacker immediately. Such attacks are harmful to every web application which receives student information and goes with it to a main database based on SQL request.

In this paper, first we give the latest attacks of SQL Injection. In Section 2 we present the literature review about the SQLIA. In Section 3 we present the SQL injection attack. Section 4 discussed about the impact of SQLIA. In Section 5 we discuss about the different types of SQL injection. Section 6 discuss about detection and prevention of SQLIA in ASP.NET application. Finally, section 7 discusses conclusion.

II. LITERATURE REVIEW

Nasar et. al [34] explores how distinct NoSQL database can efficiently manage a heterogeneous and large quantity of IoT information to satisfy the growing requirements on load and efficiency. In this paper author explain about the different time series data base for NoSQL for best suit in IoT application.

Kausar et. al. [4] proposed a system with static and dynamic investigation. In the static investigation phase, the prevention method characterizes in three level stages Malicious Text Detector, Field Constraint Authentication and Static Query Length Authentication. The proposed system uses CHECKSUM which generates small amount of code to store in database. In runtime validation phase, the client input information is approved with every one of these stages and results the client input as safe or risky. This proposed system implemented in .NET based web applications and are able to prevent almost all type of SQL Injection Attacks. This system could effectively distinguish all assaults as SQL Injection Attacks, while enabling every single query to be performed.

Author [23] proposed a new technique for detecting and preventing SQLIA. This technique is a mixture of static and dynamic approach. The technique suggested is made up of three stages. The first stage is static, all database tables to be upgraded to include a record that has only a few symbols in that stage. The second stage is dynamic, in this stage the author has suggested an algorithm to be developed once but configured for any request to be performed. The algorithm is intended to be independent of language and can therefore be used in any language. The third stage is the processing stage, the algorithm processes and monitors the execution procedures of all the queries obtained during this stage. The result of this monitoring will automatically match a set of anticipated algorithm outcomes, and the output of this comparison method will determine whether any sort of SQLIA exists.

Mahapatra et. al. [24] defined several processes such as defensive coding methods, input sanitization, to safeguard a web application from SQL Injection assault.

SQL injection attack research can be widely divided into two fundamental classifications: approaches to vulnerability detection and approaches to attack avoidance. The former category includes techniques that identify sensitive locations in a web application that could lead to SQL injection attacks. A programmer frequently submits input validation and filtering routines to all inputs to avoid SQL injection attacks, either detecting attempts to inject SQL commands or making the input benign [25].

Dubey et.al. [26] has developed a new method by combining the two existing methods that are highly secure in nature. A hash value is used in the first method to uniquely define information and has a defined length. Because the numbers will be in numerals, they have a unique function to display enormous information in much lower values. It uses hash algorithm. Proxy server is used in the second technique to protect the database against an SQL injection attack.

This will double check the login process and guarantee that the database is not entered by any intruder.

Prokhorenko et al. [27] offers a novel technique for web applications to protect against various kinds of assault. This work provides a single generic solution for different kinds of web-related injection attacks. The authors took an alternative perspective of the vulnerability's key root. In this job, the prevalent attack characteristics are evaluated and on this grounds a context-oriented model for web application security is established. But the model may not detect a backdoor presence in the code. The method may not be able to function as intended in the case of code obfuscation, code hiding, and so forth.

The generic algorithm suggested by Natarajan et al. [28] is significant in scrutinizing its easy detection mechanism against attacks by SQL injection. Testing SQL injection attack internet apps is an important step in maintaining their efficiency and quality. The suggested algorithm works much quicker and has the ability to solve SQL injection attacks with a competent solution. The paper work has been evaluated using different techniques of identification and the suggested technique can also be used on any apps that interact with databases.

III. SQL INJECTION ATTACK

In 1998, SQL injection was coined. It was registered its first use in 2000. SQL injection happens when the developer uses the hacker-connected dynamic queries to malicious variables. Bad input validation and weak coded web applications in SQLIA can benefit hackers [5]. Due to the efficient execution of SQLIA, data integrity and confidentiality are lost, leading in the loss of market value of an organization.

2.1 Reasons behind SQLIA

Nowadays, web applications are targeted the most due to its vulnerability to the attacks. There are so many reasons behind SQLIAs [6]. Due to the poor input validation property of web applications, flexibility property of SQL language and weakness in software of the web script, it gets easy for the attacker to perform SQLIA. Various reasons of SQL injection are input which are not validated, generous privileges, uncontrollable variable size, error message, client side only control, stored procedure, into out file support, and sub-select (inserting SQL query in WHERE clause of other SQL query). The major reason of SQLIA is the lack of strong input validation.

2.2 SQLIA Process

SQL injection is an attack in which, by adding it to the input parameters, the intruder inserts malicious SQL query into the web application. During SQLIA, the malicious SQL query will be inserted into the web application and the legitimate query will be concatenated. The malicious request will be executed when this request is sent to the execution database management system and SQLIA will be executed. As a consequence, sensitive information can be extracted, modified and executed in the database.

2.4 Mechanisms of SQLIA

SQLIA works on various procedures with different goals. Mechanisms of SQLIA are divided into two types, namely injection mechanism and attack intent. Injection mechanism is the procedure which is used to inject malicious code into the applications. Attackers use different types of variables and sources to inject the malicious code. This procedure further includes many categories which are used to perform the attack. First type of injection mechanism is injection through user input where user inputs the malicious data into the web application through http, ‘get’ or ‘post’ method. Second method is injection through cookies, which includes building SQL queries by using cookies content. Third method is injection through server variable where http, network headers and environmental variables are used to perform the attack. Attack intent tells about the type and the goal of various attacks. It has further many types. Identifying injectable parameter is a mechanism which discovers about the vulnerable parameter in the database.

Determining the database schema is a procedure where attacker discovers the database schema. This type of mechanism shuts down the database to deny the user. Extracting data mechanism is the method where data values are extracted which is sensitive and highly desirable. Performing denial of service mechanism shuts down the database to deny the user. Detection evading is to avoid from security mechanisms. Remote command execution includes execution of remote commands and stored procedure in the database.

IV. IMPACT OF SQL INJECTION

Due to the breakdown of confidentiality in the data stored in the databases, several of the prominent SQLIA have been impacted by information security. The main instant implications of a successful compromise are this loss of confidentiality and the resulting economic expenses for retrieval, downtime, financial penalty, and negative advertising. Table 1 depicts the Impacts of successful SQLIA.

Table 1. Impacts of successful SQLIA

Impacts	Explanation
Authentication Bypass	This attack enables SQLI attackers to navigate a database layer without offering a right username or password, potentially with administrator’s privileges.
Information Disclosure	This attack enables SQLI attackers to obtain confidential data which is stored in a repository such as data about credit cards.
Compromised Data Integrity	This attack makes it possible for SQLI attackers to alter the web page content. The result would be that a webpage is defaced.
Compromised Availability of Data	This attack enables SQLI attackers to extract data so that stored data in a database is damaged.

Remote Command Execution	This attack allows SQLI attackers to execute commands via a database that allows the attacker to regulate the operating system.

SQL injection attacks were associated with infringement of several high-profile information as seen in Table 2 below. Table 2 shows SQL injection attack from year 2009 to 2019.

Table 2. SQLIA Attack from 2009 to 2019

Year	Example
2009	Using the SQLIAs [29], the location of the Portuguese, Kaspersky and F-Secure websites of BitDefender was hacked.
2010	I. With automated SQLIA, half a million websites are struck. II. SQLIA [30] has assaulted the Royal Navy website.
2011	I. Web application firewall supplier Barracuda, infringed by SQL Injection. II. Sony is infringed by an automated SQL injection assault: Sony Canada, Sony Music Japan, Sony BMG Greece, Sony Music Portugal and Sony Pictures France. III. MySQL owned oracle has compromised its website [31].
2012	I. SQLIA attackers attack the following locations for dinner: eHarmony, Yahoo,LinkedIn, Android Forums, Last.fm, Formspring, Nvidia, Billabong, and Gamigo. II. Nvidia has recognized that SQLIA attackers have swipped up to 400,000 user profiles [30].
2013	SQL Injection Discovered at the Bangladesh Islamic Bank [32] website.
2014	Researcher discovers vulnerability to SQL injection on Tesla's website.
2015	Unidentified attackers were using SQL injection for enterprise network access [11].
2016	Crooks have been using Hack Drupal Sites SQL Injections and Fake Ransomware [12].
2017	Multiple vulnerabilities affect EMC goods, such as SQL injection [12].
2018	SQL Injection Security issues in Seagate Personal Cloud Media Server enable personal data to be retrieved [12].
2019	SQL Advance Contact Form 7 DB Injection [12].

V. TYPES OF SQL INJECTION

We present and discuss the various types of well-known SQLIAs in this section.

Tautologies

The overall goal of a tautological attack is to inject code into one or more conditional statements in order to always be assessed as true [8]. The impact of this attack depends on how the ASP.NET application used the query outcomes. Bypassing authentication websites and extracting information are the most popular uses. The attacker utilizes an injectable field in this sort of injection which is conditionally used in the "WHERE" clause. Converting the declaration into a tautology outcome in returning all the rows targeted by the query in the database table. In particular, an attacker will consider not only the injectable criteria, but also the coding concepts that assess the outcomes of the request for a tautology-based attack to work. Typically, the attack is effective if the code whether shows all the documents returned or if at least one record is returned, it conducts some action.

Example: An attacker submits "' or 1=1--" for the login input field in this instance attack (the input presented is meaningless for the other areas). The query that results is:

```
SELECT uid, password FROM user_detail WHERE uid='' or 1=1 -- AND password=''
```

The conditional code injected (OR 1=1) turns the whole WHERE keyword into a tautology. The database utilizes the conditional as the basis on which to evaluate each line and decide which ones to return to the application. Since the constraint is a tautology, the query analyzes and returns all of them to true for each row in the table. The returned result set calculates to a not null value in our instance, which leads the ASP.NET application to confirm that the user verification has been effective. The request therefore displays all the uid and password in the database return set.

Logically Incorrect Queries

This attack allows an attacker to collect significant data about a web application's database type and structure [8]. The attack is regarded to be an initial phase of collecting data for many other attacks. The vulnerability [5, 33, 6] manipulated by this attack is often overly descriptive of the default error page produced by application servers. Indeed, the easy truth that error reports are produced will sometimes show an attacker's injectable parameters. An attacker attempts to inject queries that trigger a syntax, type conversion, or logical error in the database when performing this attack. Logical errors usually show the names of the error causing tables and columns.

Example: The objective of this attack is to trigger a conversion type error which may disclose appropriate information. For this purpose, the attacker injects the following query into the password entry field: "convert (int,(select top 1 name from sysobjects where xtype='u'))". The resulting query is:

```
SELECT uid, password FROM user_detail WHERE uid='' AND password=convert (int,(select top 1 name from sysobjects where xtype='u'))
```

The injected select request tries to obtain the first user table (xtype='u ') from the metadata table of the database in the assault sequence (sysobjects is a table in SQL Server). The query then attempts to transform the name of this table to an integer. Because this is not a transformation of a legal sort, the database is making a mistake. The SQL Server bug would be: "Microsoft OLEDB Provider for SQL Server (0x80040E07) Error to convert the nvarchar value 'Registration' to an int column of data type."

There are two useful information sets in this message that help an attacker. First, since the error message explicitly states this fact, the attacker can see that the database is in SQL Server. Second, the message of error shows the value of the string that led to the transformation of the form. This value is also the name of the database's first user-defined table in this case: "Registration."

Union Query:

This kind of attack, while injecting the SQL Query, utilizes Union Operator (U). The Union Operator is entered by the two Sql query. The first declaration is a standard request that will be attached to the suspicious request with the union operator. It is therefore used to avoid the system's mechanism of prevention and detection. The instance demonstrates how this could be done.

The instance demonstrates that the 2nd query is malicious and the following text (-) is ignored because it becomes the SQL Parser's remark. The assailant attacks the web application with that kind of request to take this benefit.

```
select * from user_detail where uid='786' UNION select * from Registration where user='kausar'--' and pass='marya'
```

Piggy-backed Query:

Piggy-backed queries is a form of attack that injects extra query statements into the initial query by compromising a database using a query delimiter such as "-" The first query is authentic in this technique while the preceding queries are ingested. This attack is very damaging; the attacker can use it to inject nearly any SQL command. The SQL query below demonstrates the Piggy-backed query attack [7].

```
select Name,Address,Mobile from Registration where id = ''kausar'' AND password = 'marya'; DELETE FROM accounts WHERE Cust_Name ='Kausar';
```

The interpreter considers the ' - ' Semi Colon after executing the first request and performs the second request with the first request. The second request is malicious, so the customer's ' Kausar ' information will be deleted.

Stored Procedure:

This sort of SQLIAs attempt to execute stored procedures in the database. In a relational database management system (RDBMS), database stored procedures are commonly used as a subroutine. They are collected onto a single query optimizer and then used to execute regularly happening tasks widely. It is used in companies as it offers a single point of command during the execution of business rule [9]. IT Experts believe that Stored Procedures is a cure for SQL Injection as Stored Procedures are inserted in the front of the databases which cannot be introduced to the safety characteristics. The stored procedures cannot use the normal SQL, they use their own scripting languages that do not have the same vulnerability as SQL, but there are still distinct vulnerabilities associated with the scripting language. Given below is the example of Stored Procedure:

```
CREATE PROCEDURE User_Data
  @name varchar2 @password varchar2
  @id int
  AS
  BEGIN
  EXEC('Select user_info from User_table where name=' ''+@name '' ' and password = ' ''+@pass '' '
  GO
```

The SQL Injection Attack is susceptible to this type of processes. Any malicious customer can enter in the username and password areas the malicious information. The user's easy command may ruin the entire database or cause service disturbance.

Inference

The request is altered in this assault to recast it within the form of an intervention that is performed based on the response to a true / false information value issue in the database. In this form of assault, attackers usually attempt to attack a site that has already been sufficiently protected to prevent available feedback via database error messages when an injection has been successful. Because database error messages are insufficient to include feedback to the attacker, attackers have to use a distinct technique to get a reply from the database. The attacker inserts orders into the web site in this scenario and then explores how well the website's activity / request changes. The attacker can conclude not only whether some variables are susceptible, but also extra data about the values in the database by closely noting when the site acts the same and when its conduct varies. There are two famous inference-based attack methods. They enable an attacker to obtain and detect sensitive variables out of a database [8].

Example:

```
http://dani/productlist.aspx?cate=shoe
' or '1' = '1'.
  select * from products_details where type = 'shoe' OR '1' = '1';
```

Alternate Encodings:

This sort of assault happens when attackers use alternative encoding, including hexadecimal, ASCII, and Unicode to alter the injection request. The attacker can thus flee the developer's filter that scans input queries for specially recognized "poor personality." When this sort of attack associates with other attack methods, it might be powerful since it can reach distinct layers in the implementation, so developers need to be acquainted with each of them to offer efficient protective coding to avoid alternative encoding attacks [10]. The SQL query defines the alternative encoding attack, as shown below.

```
SELECT username,password FROM user_detail WHERE username='kausar';
exec(char(0x73687574646f776e)) -- AND password=''
```

This instance uses the function char() and the hexadecimal encoding of ASCII. The function char() requires an integer or hexadecimal encoding of a character as a parameter and returns a character. The number stream in the second part of the injection is the "SHUTDOWN" string's ASCII hexadecimal encoding. Consequently, if the request is processed by the database, the SHUTDOWN command would be executed by the database.

VI. DETECTION AND PREVENTION OF SQL INJECTION

This section presents various techniques to detect and prevent SQLIA. The techniques which are analyzed in current research are AMNESIA, SQLCHECK Approach, CANDID, Auto-mated Approach, WASP, Swaddler, Tautology Checker, WebSSari and Ardilla.

In [14] the author has created a method called AMNESIA, which accounts for SQLIA neutralization evaluation and tracking.

AMNESIA incorporates dynamic and static web applications. Vulnerability detection and prevention analysis at runtime. AMNESIA generates distinct kinds of query statements using static analysis. In the dynamic stage, before being sent to the database, AMNESIA interprets all queries and validates every request towards the statically constructed models.

SQLCHECK Approach [15] presented a technique which used grammar called “SQLCHECK” to test, validate and cleanse query. In this approach, an algorithm is implemented with the help of SQLCHECK tool, and developer applies a check on input queries. Secret key is used during parsing to restrict a user input. Only legal query is examined during runtime. Queries rely on model specified by the developer of the SQLCHECK.

In [16], the author introduces a SQLIA Automatic Prevention Dynamic Candidate Assessment Procedure called CANDID. This approach dynamically collects the query structures that are designed by the developer from each SQL query place. Therefore, CANDID solves the issue of manually modifying the application to produce the prepared statements. This technique's disadvantage is partly stopping SQLIAs due to the fundamental strategy limitations.

Automated Approach [17] highlight the use of automated approaches. There are two main schemes. First one is static analysis to find bugs approach. It detects bugs in SQLIA and gives warning when finding any malicious query or malicious variable. Second one is web vulnerability scanning in which software agent scans the web application and detects the vulnerability.

In [18] the authors created a WASP (Web Application SQL Injection Protector) tool which is effective and effective in preventing more than 12,000 assaults in non-real-time environments without producing false positives. This tool's restriction can be based on implementing the binary applications strategy and deploying web applications.

In [19], a method called Swaddler was suggested by the author to evaluate the inner status of a web application. It operates on the basis of single and multiple variables and demonstrates a remarkable manner to web applications toward complicated assaults. Initially, the strategy defines the normal values in the crucial points of the parts of the application for the state variables of the application. Later, the strategy controls the execution of the implementation to recognize unusual states during the detection stage. The limitation of this strategy is that due to some restriction of the underlying strategy it partly detects SQLIAs.

Author [20] presented a technique using static analysis and automated reasoning in order to guarantee that the input does not contain a tautology. This approach uses static analysis framework to detect the SQLIAs. There are mainly three steps which detect the SQLIA (abstract model, syntactic structure and security checking).

WEBSSARI. WebSSARI stands for Web Application Security by Static Investigation and Runtime Review. An automated tool is used in this approach which works on the basis of certified input. This input is certified by passing through the filter. It uses static analysis. In event of exposure to threat it gives an alert [21].

In [22], the authors suggested using the SQLI + XSS attack detection tool Ardilla. It has two methods to verify the assault validity, i.e. severe mode and moderate mode. Ardilla Tool utilizes Taint-based strategies and techniques of static investigation, throughout this case, if the prerequisites were not met, filters and other techniques of sanitization will be suggested in order to fulfill the prerequisite for detecting a vulnerability.

The excellent news is that it is a comparatively easy thing to do to avoid your ASP.NET application from being prone to SQLIA. Before using it in a query declaration, you must filter all user inputs. Many forms can take on filtering.

1. If you use queries that have been dynamically constructed, use the following techniques:

Delimit single quotes by exchanging a single quote instance with two single quotes to prevent changing the SQL command by the attacker. Using the instance above, `"SELECT * from login_detail WHERE uid = ''' or ''3''='3' AND pass = ''' or ''3''='3'"` has a different result than `"SELECT * from login_detail WHERE uid = '' or '3'='3' AND pass = '' or '3'='3'"`.

- Eliminate hyphens from input to avoid the attacker from creating a request comparable to: SELECT * from login_details WHERE login = 'mak ' AND password = ‘’ which would result in the second half of the request being pointed out and overlooked. It would permit an intruder to obtain access without understanding the password of the customer who understands a valid customer login.
- Restrict the user account database access rights in which the query would be executed. Use various user accounts to select, insert, update, and delete information. By dividing the activities that can be done by distinct accounts, you eliminate the chance of a declaration being inserted, updated, or deleted instead of a selected declaration or vice versa.

2. Set up and run as stored procedures all queries. The passage of SQL parameters protects apostrophes and hyphens from being used in a manner that could permit an injection attack to happen. Furthermore, it enables the restriction of database permissions so that only particular processes can be performed. All client input must at that point fit into the call procedure context and an injection attack is less probable to happen.
3. Limit the input form length or query string. If your login is 9 words long, ensure that you should not permit the value to be input with more characters than that. This will render injecting possibly damaging SQL statements into the input more hard.
4. To check that the input is limited to desired values, validate the user input. Data validation on both the client and the server should be conducted. The server side validation is needed to prevent a safety vulnerability due to the client side validation. An intruder can access and save your source code, change your validation scripts (or simply remove them), and send the form to your server with inappropriate data.

The server side validation is needed to prevent a safety vulnerability due to the client side validation. An intruder can access and save your source code, change your validation scripts (or simply remove them), and send the form to your server with inappropriate data. If, within the range of available validators, you do not find one that meets your needs, you may develop your own using the CustomValidator.

5. Store information in an encoded format like user logins and passwords. Encrypt user input to the information stored in the database for comparison. The data is now being contrasted in a sanitized way that has no importance for the database and prevents SQL instructions from being injected by the attacker. The System.Web.Security.FormsAuthentication class has a particularly useful HashPasswordForStoringInConfigFile for user input sanitization.
6. Validate the number of rows returned from a request for data recovery. If a single row of information is expected to be retrieved, then throw an error if numerous rows are collected.

VII. CONCLUSION

Today, web applications have become an important and essential component of online activity, providing the comfort of corporate and personal operations somewhere. SQLIAs, however, present a major threat to web applications. In this paper, as well as many research scientists, we have tried to address the SQL Injection Attack, which is less recognized to the general world. They are quite normal attacks on ASP.NET web applications. We addressed the methods of preventing and detecting these attacks that we might discover and apply to avoid these attacks. Due to fewer studies on SQLIA, the mitigation and detection methods mentioned are restricted. These attacks can resolve past methods of identification and avoidance and validation can be used properly. Therefore, sometimes correct Web Application coding has very little importance as it can readily be resolve. The web developer must be able to destroy the web application with the excellent understanding of these types of assaults and their implications can detrimentally effect on organization.

REFERENCES

1. Akamai, State of the Internet/Security, Q3 2017 Report.
2. Verizon, 2017 Data breach investigations report, 10th edition.
3. M. Howard and D. LeBlanc, Writing secure code. Pearson Education, 2003.
4. Kausar, M. A., & Nasar, M. An Effective Technique for Detection and Prevention of SQLIA by utilizing CHECKSUM Based String Matching. International Journal of Scientific & Engineering Research Volume 9, Issue 1, January-2018
5. Anley C. Advanced SQL injection In SQL server applications. Next Generation Security Software Ltd; 2002. p. 1–25.
6. McDonald S. SQL injection: modes of attack, defense, and why it matters. GovernmentSecurity.org; 2002 Apr. p. 1–32.
7. Alazab, A., & Khresiat, A. (2016). New strategy for mitigating of SQL injection attack. International Journal of Computer Applications, 154(11).
8. Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In Proceedings of the IEEE International Symposium on Secure Software Engineering (Vol. 1, pp. 13-15). IEEE.
9. Singh, J. P. (2017). Analysis of SQL Injection Detection Techniques. Theoretical and Applied Informatics, 28(1-2), 37-55.
10. Alwan, Z. S., & Younis, M. F. (2017). Detection and prevention of SQL Injection attack: A survey. vol, 6, 5-17.
11. breach at Pacnet, Allie Coyne May 20 2015, https://www.itnews.com.au/news/telstra-reveals-large-scale-security-breach-at-pacnet-404217 [Access Date 23.7.2019]
12. SQLi Hall-of-Shame, https://codecurmudgeon.com/wp/sql-injection-hall-of-shame/ [Access Date 23.7.2019]
13. Web Attacks and Gaming Abuse Report: Volume 5, Issue 3
14. Halfond, W. G., & Orso, A. (2006, May). Preventing SQL injection attacks using AMNESIA. In Proceedings of the 28th international conference on Software engineering (pp. 795-798). ACM.
15. Su, Z., & Wassermann, G. (2006, January). The essence of command injection attacks in web applications. In Acm Sigplan Notices (Vol. 41, No. 1, pp. 372-382). ACM.
16. Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2010). CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. ACM Transactions on Information and System Security (TISSEC), 13(2), 14.
17. Junjin, M. (2009, April). An approach for SQL injection vulnerability detection. In 2009 Sixth International Conference on Information Technology: New Generations (pp. 1411-1414). IEEE.
18. Halfond, W., Orso, A., & Manolios, P. (2008). WASP: Protecting web applications using positive tainting and syntax-aware evaluation. IEEE Transactions on Software Engineering, 34(1), 65-81.
19. Cova, M., Balzarotti, D., Felmetsger, V., & Vigna, G. (2007, September). Swaddler: An approach for the anomaly-based detection of state violations in web applications. In International Workshop on Recent Advances in Intrusion Detection (pp. 63-86). Springer, Berlin, Heidelberg.
20. Wassermann, G., & Su, Z. (2004, October). An analysis framework for security in web applications. In Proceedings of the FSE Workshop on Specification and Verification of component-Based Systems (SAVCBS 2004) (pp. 70-78).
21. Huang, Y. W., Yu, F., Hang, C., Tsai, C. H., Lee, D. T., & Kuo, S. Y. (2004, May). Securing web application code by static analysis and runtime protection. In Proceedings of the 13th international conference on World Wide Web (pp. 40-52). ACM.
22. Kieyzun, A., Guo, P. J., Jayaraman, K., & Ernst, M. D. (2009, May). Automatic creation of SQL Injection and cross-site scripting attacks. In 2009 IEEE 31st International Conference on Software Engineering (pp. 199-209). IEEE.
23. Ghafarian, A. (2017, July). A hybrid method for detection and prevention of SQL injection attacks. In 2017 Computing Conference (pp. 833-838). IEEE.
24. Mahapatra, R. P., & Khan, S. (2012). A Survey Of Sq1 Injection Countermeasures. International Journal of Computer Science and Engineering Survey, 3(3), 55.
25. C. Anley. Advanced SQL Injection In SQL Server Applications. White paper, Next Generation Security Software Ltd., 2002.
26. Dubey, R., & Gupta, H. (2016, September). SQL filtering: An effective technique to prevent SQL injection attack. In 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 312-317). IEEE.
27. V. Prokhorenko, K. R. Choo, and H. Ashman, “Context oriented web application protection model,” Applied Mathematics and Computation, vol. 285, pp. 59–78, 2016.
28. Natarajan, K., & Subramani, S. (2012). Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks. Procedia Technology, 4, 790-796.
29. Sood, A. K. (2009). The crux and the myth—breaches in security vendor websites. Computer Fraud & Security, 2009(7), 11-13.
30. D. Hartley, "Chapter 1 - What Is SQL Injection?," in SQL Injection Attacks and Defense, ed Boston: Syngress, 2012, pp. 1-25.
31. Tankard, C. (2011). Advanced persistent threats and how to monitor and deter them. Network security, 2011(8), 16-19.
32. Softpedia. (2013, April). Stories about: SQL Injection, XSS Vulnerabilities Found on the Site of Islami Bank Bangladesh, https://news.softpedia.com/news/SQL-Injection-XSS-Vulnerabilities-Found-on-the-Site-of-Islami-Bank-Bangladesh-318899.shtml, [Access Date: 26.7.2019].
33. Litchfield, David. "Web application disassembly with ODBC error messages." Windows Security (2001).

34. M Nasar, MA Kausar (2019) "Suitability of Influx DB Database for IoT Applications" 'International Journal of Innovative Technology and Exploring Engineering', 8(10),pp 1850-1857.

AUTHORS PROFILE

Dr. Mohammad Abu Kausar is working as Assistant Professor in Department of Information Systems at University of Nizwa, Sultanate of Oman. He is having more than nine years of teaching, research and Software Development experience. He has published several research papers in various International journals of repute.

Dr. Mohammad Nasar is working as Assistant Professor in Department of computing and Informatics, Mazoon College, Muscat, Sultanate of Oman. He is having more than ten years of experience in Teaching and Research. He has published several research papers in different international Journals of repute.

Dr. Aiman Moyaid is an assistant professor at the College of Economics, Management & Information Systems (CEMIS) at University of Nizwa. Aiman received his Bachelor of Information Technology & Computer Sciences from Yarmouk University, Jordan, in 2007, Master in Information Technology from Universiti Teknologi PETRONAS and his Ph.D. in Information Technology from Universiti Teknologi PETRONAS in 2014. Aiman`s main research interests include Data Mining, Data Stream Mining, Text mining, Outlier Detection.