Protecting Web Applications from SQL Injection Attacks - Guidelines for Programmers

Master Thesis Project 15 ECTS Credits
Spring 2018

Submitted By:
Gopali Gopali
garggopali@gmail.com

16/05/2018
Protecting Web Applications – Guidelines for the programmers.

Contact Information

Author:
Gopali Gopali
E-mail: garggopali@gmail.com

Academic Supervisors:
Dr. Annabella Loconsole
E-mail: Annabella.Loconsole@mah.se
Malmo University, Department of Computer Science

Dimitris Paraschakis
E-mail: dimitris.paraschakis@mah.se
Malmo University, Department of Computer Science

Industrial Supervisor:
Ross W Tsagalis, Swedish Armed Forces (SwAF)
E-mail: wross@tele2.se

Examiner:
Dr. Johan Holmgren
E-mail: johan.holmgren@mah.se
Malmo University, Department of Computer Science
Abstract

Injection attack is the most critical web application security risk, and SQL-injection (SQLi) attack is the most reported injection attack on web applications. In this thesis, we have identified the attacking techniques used by attackers and we are also providing guidelines so that the programmers can write web application code in a secure way, to prevent the SQLi attacks.

The methodology applied for the research is literature study and we used the way proof by demonstration to get the clear picture. The first step was to find out the coding flaws, then we designed guidelines that can help to protect web applications from SQLi attacks. This thesis will help the programmers to understand the various coding flaws and how those coding flaws can be prevented and for this, we have used proof by demonstration. This thesis will also contribute to the general awareness of SQLi attacks, attack types and guidelines for the programmers who are designing, developing and testing web applications.

**Keywords**: Cyber Security, Top vulnerabilities, SQL-Injection (SQLi), SQLi attack, SQLi coding flaws, Web application attacks, Code Injection cyber-attacks.
Acknowledgment

First and foremost, I am grateful to the God for the good health and well-being that were necessary to complete this master thesis.

This master thesis was written at Malmö University. I would like to thank my academic supervisors Dr. Annabella Loconsole and Dimitris Paraschakis, who helped me a lot at every step of the thesis. This study would not have been possible without the guidance, thoughts, suggestions and feedbacks from my supervisors. I would also like to thank my industrial supervisor Ross W Tsagalidis, Program manager at the Swedish Armed Forces (SwAF), for his valuable opinions, thoughts and suggestions all along the way. Furthermore, I want to thank opponents during the work with this master thesis.

My grateful thanks also extended to my examiner Dr. Johan Holmgren for his valuable comments, critiques and useful suggestions. This research would never have been completed without his assistance and guidelines.

I would also extend my thanks to my husband for his valuable feedbacks, supporting, encouragement and helping me with the research process.

Thanks to my parents, in-laws and brother for supporting and encouraging throughout my study.
# Table of Contents

Abstract .......................................................................................................................... 3

Acknowledgment ........................................................................................................... 4

List of Figures .................................................................................................................. 7

List of Tables .................................................................................................................... 8

Chapter 1 - Introduction ................................................................................................. 9
  1.1 Motivation ............................................................................................................... 9
  1.2 Goal ....................................................................................................................... 11
  1.3 Research Questions .............................................................................................. 11
  1.4 Contribution ......................................................................................................... 11
  1.5 Thesis Outline ...................................................................................................... 11

Chapter 2 - Research Methodologies ............................................................................ 12
  2.1 Literature Study ..................................................................................................... 13
    2.1.1 Searching ....................................................................................................... 14
    2.1.2 Obtaining and Assessing ............................................................................... 15
    2.1.3 Reading .......................................................................................................... 15
    2.1.4 Critical Evaluation ........................................................................................ 16
    2.1.5 Recording ....................................................................................................... 16
    2.1.6 Writing a Critical Review ............................................................................. 16
  2.2 Coding Flaws Prone to SQL Injection .................................................................... 17
  2.3 Guidelines for the Programmers ......................................................................... 17
    2.3.1 Introduction ................................................................................................... 17
    2.3.2 Lab Set-Up ..................................................................................................... 17
    2.3.3 Procedure ...................................................................................................... 17

Chapter 3 - Literature Study ......................................................................................... 18
  3.1 Previous Reports Related to Cyber Attacks ......................................................... 19
  3.2 Code Injection Attacks ......................................................................................... 19
  3.3 Classification of Code Injection Attacks ............................................................... 20
  3.4 Code Injection Attacks on Web Application ....................................................... 21
  Summary ...................................................................................................................... 22
4.1 Background .................................................................................................................. 23
  4.1.1 What is SQLi? .......................................................................................................... 23
  4.1.2 Latest SQLi Cyber Attacks ..................................................................................... 24
  4.1.3 Performing a SQLi Attack ...................................................................................... 25

4.2 Coding Flaws Prone to SQL Injection ......................................................................... 26
  4.2.1 Tautology ................................................................................................................ 27
  4.2.2 Illegal/ Logically Incorrect Queries ......................................................................... 28
  4.2.3 End of Line Comment ............................................................................................ 28
  4.2.4 Timing Attack ......................................................................................................... 29
  4.2.5 Union Query ........................................................................................................... 30
  4.2.6 Blind SQLi Attacks ................................................................................................ 31
  4.2.7 Piggybacked Queries/ Statement Injection ............................................................. 31

5.1 Introduction .................................................................................................................. 35

5.2 Guidelines for Programmers ....................................................................................... 35
  5.2.1 Parameterized Queries .......................................................................................... 36
  5.2.2 Stored Procedures ................................................................................................ 37
  5.2.3 Input Data Type Validation .................................................................................... 39
  5.2.4 Input Variable Length Checking ........................................................................... 39
  5.2.5 White List Filtering ............................................................................................... 39
  5.2.6 Smart Configuration of DBMS ............................................................................. 40
  5.2.7 Customized Error Message ................................................................................. 40

6.1 Introduction .................................................................................................................. 42

6.2 Results of Guidelines with Respect to Attack Types ............................................... 42

7.1 Summary .................................................................................................................... 45

7.2 Future Work ............................................................................................................... 46

References ......................................................................................................................... 47
List of Figures

Figure 2.1 Research Process Used in This Thesis .................................................... 13
Figure 4.1 SQLi End to End Flow ................................................................. 25
Figure 4.2 Login Form ............................................................................... 26
List of Tables

Table 2.1 Conducting a Literature Study (Oates, 2006) .............................................. 14
Table 2.2 Total hits using keywords ........................................................................... 15
Table 3.1 Selected Papers ........................................................................................... 18
Table 4.1 Attacking Techniques ................................................................................... 34
Table 5.1 Privileges According to Users ...................................................................... 40
Table 5.2 Guidelines for Programmers ...................................................................... 41
Table 6.1 Attacking Types and Guidelines .................................................................. 43
Table 6.2 Consolidation Chart - Attacking Types versus Guidelines ......................... 44
Chapter 1 - Introduction

1.1 Motivation

Cyber security has become an important and interesting topic. Poorly designed web applications may include, improper handling of requests, lack of input validation, lack of output validation, poor business-logic design & implementation. In recent years, various attacks have been carried out over the internet (Mirdula & Manivannan, 2013).

According to McGregor, (2014), the attacks on web applications are increasing day by day. The interactive map in Kaspersky Lab, (2014) shows all the current cyber-attacks occurring around the world in a real time. The website - eBay revealed that hackers had managed to steal personal records such as usernames, passwords, address and phone numbers of 233 million users using SQLi attack (McGregor, 2014). Imperva, reported an increase of 10% in SQL Injection (SQLi) attacks as compared to previous year figures (Imperva, 2013), (Imperva, 2014).

Web sites, as they provide necessary data as well as store critical information such as company statistics, credentials of users, financial and payment information etc. have been continually targeted by attackers to acquire monetary gain. SQLi and Cross Site Scripting Attack (XSS) are perhaps a few of the most common attack techniques used by attackers to manipulate or delete the content through inputting unwanted command strings in the coding part. The attacker attacks in such a way that the code manipulated will look as the original code (Johari & Sharma, 2012).
Protecting Web Applications – Guidelines for the programmers.

Unfortunately, the presence of security weaknesses in web applications allows attackers to find security vulnerabilities and become the reason behind the failure. Research on vulnerability detection, defensive programming, and attack prevention techniques has been done in the past decade (Tian et al., 2004). The aim of these approaches is to identify the weaknesses in source code before its execution in the actual environment (Jovanovic et al., 2006).

Most of the companies use anti-virus software and other security measures to prevent cyber-attacks. There is, however, a significant difference in risk mitigation costs. A recent U.S. study found that large enterprises allocate on average 12% of their IT budget on cyber security (PartnerRe, 2014).

Based on the literature study, it seems there is not enough research done at coding flaws level. Hence there is a strong need to provide guidelines for the programmers to protect web applications from the attacks. Many organizations spent a lot of money on anti-virus, data leakage prevention systems and network firewalls, but if programmers follow the proper guidelines, then organizations can save money on cyber-attack prevention. The coding errors are the most critical and widespread errors that can lead to serious vulnerabilities/hazard in software. These are often easy to find and exploit. These are very dangerous because they allow the attackers to steal data. This thesis work will guide the programmers to write code in a more secure way, in order to protect web applications from the cyber-attacks.
1.2 Goal
The goal of this research is to give the details of the SQLi attack, identify the attacking techniques and write guidelines for the programmers, so that they can write code in a secure way, to prevent the SQLi attacks.

1.3 Research Questions
In this thesis, we will address the following research questions:

1. What are the various coding flaws behind the SQLi attack?
2. How can we help programmers to prevent the SQLi attack?

1.4 Contribution
The result will be the guidelines for the programmers on how to write code to prevent SQLi attack and this will be done via a framework, listing the coding flaws with reference to the suggestions. This thesis contributes with a new perspective that gives insight into the problem of SQLi and presents results of how companies and organizations can handle these problems at code level.

1.5 Thesis Outline
This thesis is organizes as follows. Chapter 2 presents the research methodologies being used to conduct this thesis. Literature study and proof by demonstration way are discussed in this chapter. Chapter 3 presents the details of conducted literature study to present the previous work, code injection attacks and their classification. Chapter 4 discusses the background details of SQLi attacks, SQLi coding flaws. Chapter 5 is evaluation of proof by demonstration. Chapter 6 concludes the results of the guidelines with respect to attack type and Chapter 7 draws the final conclusion and ideas on the future work.
Chapter 2 - Research Methodologies

This chapter describes and motivates the choice of research methodology and how we will analyze the data. This thesis will use the research method - literature study. We will use the proof by demonstration way, in order to get an overall picture and to contribute to the general awareness.

Literature study will be used to study the major cyber-attack(s) in order to identify the vulnerabilities of the web applications and further to identify the coding flaws which lead to those vulnerabilities. Once, we identified the coding flaws, proof by demonstration way will be used to test the suggested solutions as in figure 2.1.

Research methodologies such as survey and design & creation are not recommended for this thesis. The research methodology - Survey is intended to use when we are going to collect some data from the participants, but we do not need to collect any participant data to solve any of our research questions (RQ). As per the thesis goal, we need to identify the attacking techniques used by the attackers and it could be difficult to achieve through survey. Further, approaching attackers for the survey was also very challenging. Instead, we are using literature study to study the SQLi attacks and to find the coding flaws. We are not creating any new artifact that is why the design and creation method is not applicable for this study. Definitely, it can be used in future. Software tool can be created to identify the coding flaws automatically based on the input code. Hence, research methodology like literature study is the best suitable and preferred for our thesis. With this thesis, we will come up with major coding flaws and the guidelines to prevent SQLi attack, then design & creation can be used to implement these solutions in the future using these guidelines.
Protecting Web Applications – Guidelines for the programmers.

Figure 2.1 Research Process Used in This Thesis

2.1 Literature Study

The primary research method for this study is a literature study aimed (1) to examine the previous research studies around cyber security and the subject of SQLi attack to examine what has already been written, and (2) to increase the author’s knowledge in the subject area (3) to find out coding flaws. During the whole literature study process, we will follow the steps for conducting the review mentioned by (Oates, 2006). The seven steps are followed as searching, obtaining, assessing, reading, critical evaluation, recording, and writing a critical review which is shown in table 2.1. A literature study using Web of Science and IEEE Explore was conducted. Web of Science and IEEE Explore gives access to the leading citation databases, making it possible to search in citation databases, influential journals and in the proceedings of the conferences. Literature Study helps the researcher to collect, analyze, evaluate and organize literature in order to conduct a study. Literature search will be performed using the internet to collect the data.
2.1.1 Searching

The searching strategy started by systematically reading and scanning journals, books, conference papers, articles, published about the coding flaws that are the reasons of the attacks on the web applications. The digital database containing published papers such as ACM, IEEE, Science Direct, Springer, Elsevier and Google scholar were used to search desired articles. We accessed those articles by using the Malmö University digital library. Internet resources were also frequently used to search articles about programming flaws. We found the published articles in between the year 2004-2017.
2.1.2 Obtaining and Assessing

The obtaining and assessing information about SQLi coding flaws comes from journal papers, articles, conference and as well as white papers. We obtained that information through Malmö university digital library which allows us to get information freely. We also obtained information by using Google search engine and assessing the source by using the Google scholar website. The search term that we used to obtain information related to our topic were Cyber Security, Top vulnerabilities, SQLi, SQLi attack, SQLi coding flaws, Web application attacks, Code Injection cyber-attacks. We found the following number of hits per keyword.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Total hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyber security</td>
<td>268 000</td>
</tr>
<tr>
<td>Top vulnerabilities</td>
<td>138 000</td>
</tr>
<tr>
<td>SQL-Injection</td>
<td>14 800</td>
</tr>
<tr>
<td>SQL-Injection attack</td>
<td>12 300</td>
</tr>
<tr>
<td>SQL Injection coding flaws</td>
<td>5 570</td>
</tr>
<tr>
<td>Web application attacks</td>
<td>589 000</td>
</tr>
<tr>
<td>Code Injection cyber attacks</td>
<td>12 700</td>
</tr>
</tbody>
</table>

Table 2.2 Total hits using keywords

We have selected about 30 articles related to the SQLi, coding flaws and cyber-attacks, from the total hits related to keywords that are mentioned in table 3.1.

2.1.3 Reading

The articles that we found related to our topic SQLi and subject areas such as coding flaws and how to prevent these attacks. For each article, we first read the abstract section; if the article was related to our topic, then we were able to obtain more information such as the related works, Methodology, Discussion and Conclusion sections of those articles. This provided us a clear overview what the researchers have contributed or either added any knowledge values or filled any knowledge gaps in the research area of SQLi.
2.1.4 Critical Evaluation

To investigate our research questions, we first read the articles that were related to our research topic. Then, we assessed articles which provided information about SQLi such as coding flaws, detection and prevention. Most of the articles provided more theoretical baseline information and technical overview of SQLi attacks rather than practical - how we can prevent these attacks. Further, it was checked if the paper coming from journal or conference or from other reliable source. Authenticity and validity of every considered source has been checked.

2.1.5 Recording

In this step, we wrote a brief summary of the content and an evaluation of the selected articles, which were published in the year 2004-2017. Some important coding flaws and coding practices that can protect from SQLi attacks were annotated. We also kept a note for the bibliographic details of the literature resources by using the google scholar web site.

2.1.6 Writing a Critical Review

To structure the collected data from the articles that were related to our research questions, we used to write a critical review of the research articles. To maintain the record, we used the keywords with their author and published year.

Based on the obtained articles published in the year 2004-2017, we also made a matrix containing the mapping between the author name, publications and the different concepts surrounding the SQLi.
2.2 Coding Flaws Prone to SQL Injection

The literature study methodology is used to search the major cyber-attack(s) to identify the vulnerabilities of the web application and further to identify the coding flaws which lead to those vulnerabilities in chapter 4.

2.3 Guidelines for the Programmers

2.3.1 Introduction

We are going to create the guidelines for the programmers on how to write code to prevent SQLi attack. To do that the literature review will give us input about the various kinds of coding flaws prone to SQLi attack. For guidelines, we will replicate the studied SQL coding flaws. To execute the proof by demonstration, test environment has been setup. Based on the proof by demonstration, we will come up with some guidelines and this will be representing via a framework, listing the coding flaws with reference to the suggestions.

2.3.2 Lab Set-Up

- Setup a test environment using programming language C# and SQL Server to do test on coding flaws as per section 4.2 according to the guidelines mentioned under section 5.2.

- This proof by demonstration was performed on the author’s personal computer.

2.3.3 Procedure

The proof by demonstration process is the following:

1. Coding flaws prone to SQLi: under this step we have used literature study to find out the coding flaws prone to SQLi. (refer to section 4.2)

2. Replication of the SQL coding flaws: under this step replication of the studied SQL coding flaws is done. (refer to chapter 5)
Chapter 3 - Literature Study

This section provides a review of the key research references used to define the most recent attacks initiated by the flaws in coding part. This section also gives the information about related work that have done related to this concept and gives comparison about different research papers that have done earlier. We have found many papers related to the keywords mentioned in table 2.2. The selected papers are listed in the table 3.1.

<table>
<thead>
<tr>
<th>Title</th>
<th>Articles</th>
</tr>
</thead>
<tbody>
<tr>
<td>A simple and efficient framework for detection of sql injection attack</td>
<td>(Win &amp; Hmun, 2013)</td>
</tr>
<tr>
<td>Seven pernicious kingdoms: A taxonomy of software security errors</td>
<td>(Tsipenyuk et al., 2005)</td>
</tr>
<tr>
<td>Arm up administrators: automated vulnerability management. In Parallel Architecture</td>
<td>(Tian et al., 2004)</td>
</tr>
<tr>
<td>CWE/SANS Top 25 Most Dangerous Software Errors</td>
<td>The MITRE Corporation, 2011</td>
</tr>
<tr>
<td>CWE/SANS Top 25 Most Dangerous Software Errors</td>
<td>The MITRE Corporation, 2010</td>
</tr>
<tr>
<td>Classification of sql injection attacks</td>
<td>(Sun et al., 2007)</td>
</tr>
<tr>
<td>SQL Injections–A Hazard to Web Applications</td>
<td>(Singh &amp; Purwar, 2012)</td>
</tr>
<tr>
<td>Automated removal of cross site scripting vulnerabilities in web applications</td>
<td>(Shar &amp; Tan, 2012)</td>
</tr>
<tr>
<td>SQL injection is still alive: A Study on SQL injection signature evasion techniques</td>
<td>(Sadeghian et al., 2013)</td>
</tr>
<tr>
<td>Defining code-injection attacks</td>
<td>(Ray &amp; Ligati, 2012)</td>
</tr>
<tr>
<td>What isn't Vulnerable to Cyber Attack.</td>
<td>(PattinRe, 2014)</td>
</tr>
<tr>
<td>Anonymous Hacker breached WTO database and Leaked data of internal staff</td>
<td>(Paganini, 2015)</td>
</tr>
<tr>
<td>Countering code injection attacks: a unified approach</td>
<td>(Mitropoulos et al., 2011)</td>
</tr>
<tr>
<td>Security vulnerabilities in web application - An attack perspective</td>
<td>(Mirdula &amp; Manivannan, 2013)</td>
</tr>
<tr>
<td>The Top 5 Most Brutal Cyber Attacks Of 2014 So Far</td>
<td>(McGregor, 2014)</td>
</tr>
<tr>
<td>CyberThreat Real Time Map</td>
<td>(Kaspersky Lab, 2014)</td>
</tr>
<tr>
<td>PiXY: a static analysis tool for detecting Web application vulnerabilities</td>
<td>(Jovanovic et al., 2006)</td>
</tr>
<tr>
<td>Web Application Attack Report #5</td>
<td>(Imperva, 2014)</td>
</tr>
<tr>
<td>Web Application Attack Report</td>
<td>(Imperva, 2013)</td>
</tr>
<tr>
<td>A classification of SQL-injection attacks and countermeasures.</td>
<td>(Halfond et al., 2006)</td>
</tr>
<tr>
<td>Hacker group claims to have looted 100k via sql injection attack</td>
<td>(Greenberg, 2013)</td>
</tr>
<tr>
<td>Up to 100K Archos customers compromised by SQL injection attack</td>
<td>(Danikewater, 2015)</td>
</tr>
</tbody>
</table>

Table 3.1 Selected Papers
3.1 Previous Reports Related to Cyber Attacks

Imperva found that SQLi attacks were up by 10 percent and Remote File Inclusion (RFI) attacks were up by 24 percent as compared to the previous few years figures (Imperva, 2013), (Imperva, 2014). According to these reports, cyber-attacks such as SQLi, XSS, RFI, Local File Inclusion (LFI), Directory Traversal (DT) still are the dangerous attacks on web applications. These attacks are still out of control and increasing day by day (Imperva, 2013) :(Imperva, 2014).

3.2 Code Injection Attacks

Code injection is a technique to introduce code into a system or a computer program by taking advantage of unchecked assumptions. The authors (Mitropoulos et al., 2011) have presented an approach that counters a specific class of code injection attacks. They propose a generic approach that prevents the class of injection attacks. Their scheme detects attacks by using location-specific signatures to validate code statements. The signatures are the unique identifiers that represent specific characteristics of a statement’s execution. They have applied their approach successfully to defend against attacks targeting SQL, JavaScript and XPath (Mitropoulos et al., 2011).
Ray and Ligatti, (2012) have presented existing definitions of code-injection attacks. The most common types of attack involve injecting code into a program by an application (Martin et al., 2011), for example an attacker is entering the string as input to an application. The statement SELECT col_name FROM table_name WHERE password='' OR 1=1 - -’ always returns the values from the table_name, even though an empty string password is supplied, because: (1) the 1=1 sub expression is true, making the entire WHERE clause true, and (2) the - - command comments out the final apostrophe to make the program syntactically valid (Ray & Ligatti, 2012).

The study in (Mitropoulos et al., 2011) is concentrated on a scheme that detects attacks by using location-specific signatures to validate code statements, but comparatively Ray and Ligatti, 2012 have presented code injection example, but in both studies (Mitropoulos et al., 2011),(Ray & Ligatti, 2012) the objective is common, both studies have presented the code injection technique.

### 3.3 Classification of Code Injection Attacks

Holm and Ekstedt, (2012) have described that the code injection attacks can be classified into source code injection attacks and binary code injection attacks. A binary code injection involves insertion of malicious code in a binary program. Source code injection attacks involve interaction with applications written in programming languages such as JavaScript, PHP and SQL statements. Source code injections primarily concern with web application. They referred these attack types as Web Application Injections (Holm & Ekstedt, 2012).

Tsipenyuk et al., 2005, have organized the source code errors into taxonomy. They want to teach programmers to recognize categories of problems that lead to vulnerabilities and identify existing errors as they build software. The information contained in their taxonomy is related to the coding errors (Tsipenyuk et al., 2005).
Holm and Ekstedt, 2012 concentrated on attack types such as source code attacks and binary code attacks. Instead Tsipenyuk et al., 2005, concentrated on source code attacks. In both studies (Holm & Ekstedt, 2012), (Tsipenyuk et al., 2005) the objective is common, both studies have organized sets of security rules that can be used to help software programmers to understand the kinds of errors that have an impact on security. Their taxonomy includes coding errors that occur in a variety of programming languages. The most important among them are C, Java, C++ and .NET family, including C# and ASP (Tsipenyuk et al., 2005).

3.4 Code Injection Attacks on Web Application

Attackers violate web applications vastly. This is mainly due to "improper validation of the input variables" (Mirdula & Manivannan, 2013). SQLi- Techniques makes use of code to exploit the vulnerability in a Web application. The attacker provides Structured Query Language (SQL) code injection to gain unauthorized and unlimited data access.

The attackers attack in such a way that the original code looks like an SQL code (Mirdula & Manivannan, 2013). XSS vulnerability is among the top web application vulnerabilities, according to Imperva, (2014). This vulnerability occurs when a web application uses inputs received from users in web pages without validating those (Shar & Tan, 2012). This allows any attacker to inject malicious scripts in web pages so that the scripts perform malicious attacks, when a client visits the exploited web pages. Such attacks may cause serious security violations such as cookie theft and account hijacking. The authors have described a few coding rules to remove the XSS vulnerabilities in server programs (Shar & Tan, 2012). The objective of these studies is common, both studies explain the web application attacks (Mirdula & Manivannan, 2013), (Shar & Tan, 2012).


**Summary**

Literature study has given a fair idea of what others have already studied or discovered within the field of cyber-attacks in web applications. The main result of this literature study is that SQLi is the major attack and the main reason behind is the coding flaws, hence there is a strong need to provide guidelines for the programmers, so they can protect the web applications from the SQLi attacks.

Through the literature study, we have identified the latest cyber-attacks, code injection attacks, their classification and coding flaws. The next chapters present details about SQLi attack, the coding flaws and the ways to prevent them.
Chapter 4 – SQL Injection Attack

From the literature study, we came to know that SQLi is the major cyber-attack in the web applications that is increasing day by day. Here, more details will be described about SQLi attack. To answer the research questions (RQs), we first need to find out what the coding flaws regarding the SQLi attack are and then, we need to find out the ways to prevent these attacks. The study of the latest cyber-attacks such as NextGEN Gallery, Airsoft GI, Archos smartphone, World Trade Organization (WTO) and Sebastian California has been described under section 4.1.2.

4.1 Background

In the following sections, we have described about the SQLi, their latest attacks, and the coding flaws, so that they can mitigate the SQLi attack.

4.1.1 What is SQLi?

Currently, SQLi attack is one of the most dangerous and common threats to databases and web applications. It typically involves malicious updates, modifications of the user SQL input either by changing the structure of an existing clause or by adding the additional clauses. SQLi enables attackers to access, delete or modify critical information in a database without proper authorization and become the reason of SQLi attacks (Win & Htun, 2013).

We have identified the piece(s) of code behind that coding flaw using literature study. SQLi is an attack that exploits a security vulnerability occurring in the database of an application. The attacker can extract, manipulate or delete the web application data using SQLi.
4.1.2 Latest SQLi Cyber Attacks

Here are five recent examples of SQLi attacks on companies.

1. **NextGEN Gallery** - In 2017, NextGEN is a most popular gallery plugin was attacked with SQLi to access the database that stores very sensitive user details and then the researchers said that the attackers used two scenarios to steal user data (Vatu, 2017).

2. **Airsoft GI** - In 2017, Airsoft GI is California based company. The hackers have attacked the Airsoft GI forum and stolen 65,000 accounts that include personal details of registered users. Hackers had stolen 40,000 gmail accounts, 2,500 outlook accounts, 3,000 yahoo accounts, 2,500 hotmail accounts. This attack has been initiated via SQLi (Amir, 2017).

3. **Archos Smartphone** – In 2015, Archos French Smartphone maker fell victim to an SQLi attack, and 100,000 customer records were released, containing personal and corporate email addresses, and International domains including customer’s first names and last names. This attack is done by Focus group on Twitter (Danikwater, 2015).

4. **World Trade Organization (WTO)** - In 2015, the hackers have attacked the World Trade Organization (WTO) website and leaked 53,000 member's personal data. This attack has been initiated via SQLi. The stolen data was related to internal staff of WTO which was located in various countries such as US, India, Brazil, Russia and many other countries (Paganini, 2015).

5. **Sebastian California** - In 2013, Sebastian a California-based phone, internet and television service provider company was attacked with SQLi to access the usernames and passwords of customers and then the attackers used those credentials to steal $100,000 worth data from online accounts. The attacker used the Sebastian account credentials for accessing Gmail and then was able to easily access the Google account (Greenberg, 2013).
4.1.3 Performing a SQLi Attack

The SQLi attack takes advantage of weak user input validation, to insert malicious code into the back-end database. Figure 4.1 illustrates how a SQLi can be performed step by step: 1) A user is accessing a web application by typing the address in the URL. 2) The attacker injects malicious code to the web application. 3) The malicious SQL-query is passed to the database server from the web server. 4) The database management system sends the results based on injected code back to the web server. The results can be some data or error message or confirmation, depends on injection type. 5) The web server sends/shows the same result back to the attacker. This is an example of a regular SQLi attack through user input, where the user can see and exploit an error message from the DBMS.

![Figure 4.1 SQLi End to End Flow](image-url)
4.2 Coding Flaws Prone to SQL Injection

To prevent SQLi attacks, firstly we need to know the various methods through which the attackers can find the coding flaws to invoke the attack. SQLi attacks can be invoked in the following ways: i) Tautology ii) illegal/ logically incorrect queries iii) End of line comment iv) Timing attack v) Union queries vi) Blind SQLi attack vii) Piggybacked queries (Kindy & Pathan, 2011).

To explain these attacking types, We have tried to replicate these attacking types and following example has been used:

![Login Form](image)

**Figure 4.2 Login Form**

In this case, we have username 'ced' and password '123'. When the users click on the login button then it will redirect to another page.

The coding behind the login button is described in the following example.

**Example**

```csharp
protected void Button1_Click (object sender, EventArgs e)
{
    string CS = ConfigurationManager.ConnectionStrings["DBConnectionString"].ConnectionString;
    SqlConnection con = new SqlConnection(CS);
    SqlCommand cmd = new SqlCommand ("SELECT * FROM User_Details WHERE username =" + TextBox1.Text + ' " and password="' + TextBox2.Text + '", con);
    con. Open ();
```
Here Id, username and password are the attributes of the table User_Details. Now we are going to show using various attacking types, how a malicious user can get access to this page without knowing the correct username or password.

4.2.1 Tautology

This technique injects statements that are always true, so that the queries always return results upon evaluation of WHERE condition. The login process is the most important and starting step for the attackers. Probably they want to log in with a high level of permissions. When programmers are building a login process, they simply match the records from the database. If the record does not match the values from the database then it is assumed that the user is not authenticated. If the records are matched, then the user can log in correctly. The attackers do this by adding a small text string in the username field such as OR'1'='1'-- (Kumar & Pateriya, 2012). This attack can be prevented using guidelines such as Parameterized queries as in 5.2.1, Stored procedures as in 5.2.2, Input variable length checking as in 5.2.4, White list filtering as in 5.2.5.

Example

```
SELECT * FROM User_Details WHERE username='ced' AND password='123';
```

The resulting query will be:

```
SELECT * FROM User_Details WHERE username = " OR '1'='1'-- and password='123';
```

Using this query, the attacker can get all the record from the table because WHERE clause always return TRUE value.
4.2.2 Illegal/ Logically Incorrect Queries

This method comes under manipulation category where the attacker used to collect information about the database. The attackers can inject malicious input in the query to produce some logical errors or syntax errors (Srivastava, 2014). This attack can be prevented using guidelines such as White list filtering as in 5.2.5, Customized error message as in 5.2.7.

Example: Find-out table-name and column-name

Original Query: xyz/profile.aspx/? id=5
Injected Query: xyz/profile.aspx/? id=5'

Error: With the help of this query attacker get an error message like: "SELECT username FROM User_Details WHERE id=5\""

From this error attacker comes to know about the table name User_Details and the column name is id. The error message will give more information to the attackers than they had before.

4.2.3 End of Line Comment

This is used when the attacker has some knowledge related to the database but not the full details. Then the attacker can use half detail and add -- in the end of the input field so that the rest of information is treated as a comment (Sun et al., 2007). This attack can be prevented using guidelines such as Parameterized queries as in 5.2.1, Stored procedures as in 5.2.2, White list filtering as in 5.2.5.

Example

SELECT * FROM User_Details WHERE username='ced' AND password='123';

Injected query

Name - Gopali
Master in Computer Science
4.2.4 Timing Attack

Timing attacks are often used when there is no other way to retrieve information from the database server. For this attack, the attacker injects a SQL query that generates a time delay. The attacker guesses the information character by character that depends on the output form of value as true or false. An attacker collects information by observing the response time from the database to fetch information from the application. The attacker asks questions through the queries and sets delay times for a particular condition in the query. If the condition is true, then the attackers are allowed to gain information about the application. The attackers usually get information about which database is used, database version, which field is vulnerable, user is system administrator or not (Sun et al., 2007). This attack can be prevented using guidelines such as Parameterized queries as in 5.2.1, stored procedures as in 5.2.2, Input variable length checking as in 5.2.4, White list filtering as in 5.2.5, and Smart configuration of DBMS as in 5.2.6.

Example:

Original Query:
SELECT * FROM User_Details WHERE username="ced" AND password="123";

Injected Query:
SELECT * FROM User_Details WHERE username="ced" AND ascii(substring(pwd,1,1)) > Z waitfor delay "0:0:7" -- AND password="not required";

Result- The attackers get information that, if the ASCII value of the first character of the password is greater than the value z then the query will generate a delay for seven seconds.
4.2.5 Union Query

This type of attack comes under SQL manipulation attack. This type is used by attackers to get information related to other tables. In this, the injected query is concatenated with the original SQL query using the keyword UNION (Sun et al., 2007). This attack can be prevented using guidelines such as Input data type validation as in 5.2.3, Smart configuration of DBMS as in 5.2.6.

Example - 1

**Original query:**
```
SELECT username FROM User_Details WHERE id='5';
```

**Injected query:**
```
SELECT username FROM User_Details WHERE id='5'
UNION
SELECT Price FROM tblProduct WHERE id='5'
```

**Result** - It will return price information of the table tblProduct.

Example - 2

**Original Query**-
```
SELECT username FROM User_Details WHERE id='2';
```

**Injected Query**-
```
SELECT username FROM User_Details WHERE id='
UNION
SELECT * FROM User_Details.
```

**Result** - Now, the above mentioned source code will return all the detailed information of the table User_Details.
4.2.6 Blind SQLi Attacks

In this attack, the attackers test for SQLi vulnerabilities by sending the input that would cause the server to generate an invalid SQL query and if the server returns to the client an error message, then the attacker will attempt to reverse-engineer portions of the original query using information gained from those error messages. To detect blind SQLi vulnerabilities, the attacker sends the queries to that database based on true false conditions, to see if the database server responds in the same way as it does with a normal condition or its responding something different (Kindy & Pathan, 2011). This attack can be prevented using guidelines such as Input data type validation as in 5.2.3, Input variable length checking as in 5.2.4, White list filtering as in 5.2.5, Smart configuration of DBMS as in 5.2.6, Customized error message as in 5.2.7.

4.2.7 Piggybacked Queries/ Statement Injection

In this attack, the attacker is used to inject a new SQL query into the original SQL query, through the front-end application using the query delimiter, such as “;” (Singh & Purwar, 2012). This attack can be prevented using guidelines such as Parameterized queries as in 5.2.1, Stored procedures as in 5.2.2, White list filtering as in 5.2.5.

Example - SQLi can happen because the SQL statements build dynamically, by concatenating the strings. The example below shows vulnerable code which can further become the reason behind the cyber-attack.

Example

```csharp
protected void btnGetProductByName_Click (object sender, EventArgs e)
{
    string CS = ConfigurationManager.ConnectionStrings["DBConnectionString"].ConnectionString;
    SqlConnection con = new SqlConnection(CS);
```
SqlCommand cmd = new SqlCommand("SELECT * FROM tblProduct WHERE Name = ' " + TextBox1.Text + ' '", con);
    con.Open();
    GridView1.DataSource = cmd.ExecuteReader();
    GridView1.DataBind();
    con.Close();

In this example the programmer use ("SELECT * FROM tblProduct WHERE name=' " + TextBox1.Text + " '",con) statement. It means that we want to select all columns from a specific table where column "Name" is equal to whatever the user has typed in the text box. Now the attackers can attack by entering the injected data so that the values in the database can be deleted or changed.

In the example if the users enters "Books" and clicks on "GetProductByName" then they will get the result of books like Id, name and the price of the book, but the problem will occur when the user enter some malicious code into the text box. For example, if the user will enter "Pens'; DELETE * FROM tblProduct -- " in the user text box then first, it will show the results of "Pens", but after that when user will click on "GetAllProducts",

protected void btnGetAllProducts_Click(object sender, EventArgs e)
    {
    string CS = ConfigurationManager.ConnectionStrings["DBConnectionString"].ConnectionString;
        SqlConnection con = new SqlConnection(CS);
        SqlCommand cmd = new SqlCommand("SELECT * FROM tblProduct", con);
        con.Open();
        GridView1.DataSource = cmd.ExecuteReader();
        GridView1.DataBind();
        con.Close();
    }
Then it will not show us any results, because the attacker has injected some malicious code to delete all the table entities. The attacker will successful attack and be able to delete all the entities from the table.

**Summary**

In this section we have studied the latest cyber-attacks such as Airsoft GI, GenNext gallery, Archos smartphone, World Trade Organization (WTO) and Sebastian California. We have described about the SQLi and their latest attacks. We have find out the various methods through which the attackers can find the coding flaws to invoke the attack. There are few ways through which SQLi attack can be invoked. The ways described in the chapter 4, can be used as attacking techniques. It is shown in the table 4.1(on next page).
### Table 4.1 Attacking Techniques

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Technique</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tautology</td>
<td>This technique injects statements that are always true, so that the queries always return results upon evaluation of WHERE condition.</td>
<td>Section 4.2.1</td>
</tr>
<tr>
<td>2</td>
<td>Illegal/Logically incorrect queries</td>
<td>The attacker injects malicious input in the query to produce some logical errors, in order to get the information.</td>
<td>Section 4.2.2</td>
</tr>
<tr>
<td>3</td>
<td>End of line comment</td>
<td>The attacker uses half detail and adds '--' in the end of the input field so that the rest of information is treated as a comment.</td>
<td>Section 4.2.3</td>
</tr>
<tr>
<td>4</td>
<td>Timing attack</td>
<td>The attacker collects information by observing the response time of the database.</td>
<td>Section 4.2.4</td>
</tr>
<tr>
<td>5</td>
<td>Union Query</td>
<td>The attacker injects query to concatenate with the original SQL query using the keyword UNION.</td>
<td>Section 4.2.5</td>
</tr>
<tr>
<td>6</td>
<td>Blind SQLI attacks</td>
<td>The attacker sends the database true or false questions and determines the answer based on the applications response in order to see if the db server responds in the same way as it does with a normal condition or different.</td>
<td>Section 4.2.6</td>
</tr>
<tr>
<td>7</td>
<td>Piggy backed queries/Statement Injection</td>
<td>This is used when attacker knows little bit information related to database but not full details. Then attacker can use half detail and add -- in the end of the input field so that the rest of information is treated as a comment.</td>
<td>Section 4.2.7</td>
</tr>
</tbody>
</table>
Chapter 5 - Evaluation

5.1 Introduction

We have represented guidelines via a framework, listing the coding flaws with reference to the suggestions. We identified the attacking types that are used by attackers to inject malicious code that is presented in chapter 4 using several different criteria. We first consider which attack type can be used by attackers, and then we identified how to prevent SQLi cyber-attacks using proof by demonstration.

The literature study is used to find out the major attacks in web applications that is SQLi attack and the coding flaws behind those attacks. Then proof by demonstration is used to test and evaluate some piece(s) of code. It can be used as the guidelines for the programmers. In this thesis, we will give guidelines to the programmers to prevent SQLi attack using literature study and proof by demonstration way.

5.2 Guidelines for Programmers

Here, we are describing the ways which can help the programmers to avoid SQLi attacks in the web applications. After identifying the coding flaws and piece(s) of code using various attacking types listed under the chapter 4, we have used a test environment to create GUI written in C# and SQL Server to accept the user inputs. Now, we have modified the same code to fix that coding flaw and test to see, if it is still possible to initiate the SQLi attack.
5.2.1 Parameterized Queries

Parameterized statements also known as prepared statements, can be used to construct SQL-queries in a secure and safe way. If these queries can be used correctly, then programmers can prevent SQLi attacks (Sadeghian et al., 2013). This can be used to prevent attack types such as Tautology as in 4.2.1, End of line comment as in 4.2.3, Timing attacks as in 4.2.4, Piggy backed attacks as in 4.2.7. There are two major benefits with prepared statements such as (1) The applications will work faster because parameterized queries will use the fewer resources and can be executed many times using same or different parameters, (2) The parameterized statements will automatically take care of quotes. This makes a web application safe from SQLi attacks, if used exclusively.

Example

protected void btnGetProductByName_Click(object sender, EventArgs e)
{
    string CS = ConfigurationManager.ConnectionStrings["DBConnectionString"].ConnectionString;
    SqlConnection con = new SqlConnection(CS);
    SqlCommand cmd = new SqlCommand("SELECT * FROM tblProduct WHERE Name =@Name", con);
    cmd.Parameters.Add(new SqlParameter("@Name", TextBox1.Text));
    con.Open();
    GridView1.DataSource=cmd.ExecuteReader();
    GridView1.DataBind();
    con.Close();
}

In the above example, we are doing coding part using parameterized queries. Here, if the user will enter "Books" in the text box, then the user will get the results of the Book. The attacker is trying to inject SQLi by entering the same statement as earlier "Pens";
DELETE * FROM table_name -- " , it will not do any changes into the database, because now the value in the text box is treated as the value of parameter. Here the SQLi attack is not possible. Therefore, this is the one method through which programmers can avoid SQLi attack. It shows that SQLi attack is not possible at all when we have used parameterized statements in our code.

5.2.2 Stored Procedures

Stored procedures are another solution that can help to prevent SQLi. To use this prevention technique, the programmer first defines code in SQL, and then sends the parameters into the SQL code (Elshazly et al., 2014). This can be used to prevent attack types such as Tautology as in 4.2.1, End of line comment as in 4.2.3, Timing attacks as in 4.2.4, Piggy backed attacks as in 4.2.7. There are two major benefits with stored procedures.

1) Stored procedures can reduce network traffics between the web server and the database server. If we are using statements inside stored procedures from the web server to the database server then we just need to send the name of the stored procedure, we do not need to send the lengthy SQL statements.

2) The other advantage of stored procedures is reusability, since the code is there in the form of stored procedures, then we can reuse the stored procedures on multi pages without having to duplicate. Maintenance will be much easier using stored procedures.

The example below shows the creation of store procedure instead of writing individual SQL statements in SQL server.

**Example**

```sql
Create Proc spGetProductByName
@Name varchar(30)
```
Here we are going to use a stored procedure in coding part. We specify the command name as a stored procedure.

**Example**

```csharp
protected void btnGetProductByName_Click(object sender, EventArgs e)
{
    string CS =
    ConfigurationManager.ConnectionStrings["DBConnectionString"].ConnectionString;
    SqlConnection con = new SqlConnection(CS);
    SqlCommand cmd = new SqlCommand("spGetProductByName", con);
    cmd.CommandType=CommandType.StoredProcedure;
    cmd.Parameters.Add(new
    SqlParameter("@Name",TextBox1.Text));
    con.Open();
    GridView1.DataSource=cmd.ExecuteReader();
    GridView1.DataBind();
    con.Close();
}
```

If the user will enter "Books" in the text box, then he will get the results of the Book. If the user is trying to inject SQLi by entering the same statement as earlier "Pens'; DELETE *
FROM table_name --", then the attacker is not successful to do attack, but if we will click on "GetAllProducts" then we will get all the product names. Therefore, here the SQLi attack is not possible. Hence, programmers can avoid SQLi attack by using stored procedures as well.
5.2.3 Input Data Type Validation

SQLi attack can be performed by injecting commands into either a numeric parameter or a string parameter. Even a simple input checks can prevent many small attacks. The programmers need to validate the input data types in correct manner. The programmer must define the input data types as string or numeric or any other type. If the input data entered by the end user is incorrect, then wrong inputs could be easily rejected based on the declared data types at code level. For example, in the case of numeric inputs, the programmer can simply reject any input that contains characters other than numbers. This can be used to prevent attack types such as Union query as in 4.2.5, Blind SQLi attacks as in 4.2.6. Many programmers omit this kind of validation accidentally because user input is almost always represented in the form of a string (Halfond et al., 2006).

5.2.4 Input Variable Length Checking

If the programmers add some validation checks in the program such as input variable length, then the malicious code strings beyond certain length limits will not be applicable. Even if the length limitation is long enough to fit a one or two additional queries, still the inability to input an infinitely long string can help to secure the web applications (Sun et al., 2007). This can be used to prevent attack types such as Tautology as in 4.2.1, Timing attacks as in 4.2.4, Blind SQLi attack as in 4.2.6.

5.2.5 White List Filtering

Special characters are normally used during injection attacks; therefore the programmer should need to characterize such special characters as the black list filtering. The filtering approach is suitable for the well-structured data values, such as dates, email address etc. and programmer should keep a list of data patterns and accept only matching input data (Elshazly et al., 2014). This can be used to prevent attack types such as Tautology as in 4.2.1, Illegal/ Logically incorrect queries as in 4.2.2, End of line comment as in 4.2.3, Timing attacks as in 4.2.4, Blind SQLi attack as in 4.2.6, Piggy backed attacks as in 4.2.7.
5.2.6 Smart Configuration of DBMS

A smart configuration of the database management engine can mitigate the risk of attacks. Defining different users with different privileges and using them in the development process can be helpful in mitigating the risk of SQLi attack. Use of “different accounts” or “least privileges” are also related terms (Sadeghian et al., 2013), (Sun et al., 2007). This can be used to prevent attack types such as Timing attacks as in 4.2.4, Union query as in 4.2.5, Blind SQLi attack as in 4.2.6.

In table 5.1, we defined three different users and limited their privileges. In this case, if the attacker found vulnerability in a web page then the attacker will be able to run only select commands. However selected command still can be a powerful command in hand of an attacker, but it can eliminate the risk of deleting data by the attacker.

<table>
<thead>
<tr>
<th>Users</th>
<th>Privileges assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewer</td>
<td>Select operation</td>
</tr>
<tr>
<td>Advance Users</td>
<td>Insert, Delete, Update</td>
</tr>
<tr>
<td>Administrator</td>
<td>Create, Drop, Alter, Execute etc.</td>
</tr>
</tbody>
</table>

Table 5.1 Privileges According to Users

5.2.7 Customized Error Message

Attackers may gain access through informative error messages, but if the programmers completely remove error messages, then it will make debugging a difficult task. Programmers can customize the error messages, so that the attackers cannot get information regarding the database and other important objects (Sun et al., 2007). This can be used to prevent attack types such as Illegal/ logically incorrect queries as in 4.2.2, Blind SQLi attack as in 4.2.6.
Summary

In this section we have used proof by demonstration to check if it will help to mitigate the SQLi attack. The steps described in the chapter 5 can be used as guidelines for programmers to protect the web applications from SQLi attacks in table 5.2.

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Type/Technique</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parameterised Queries</td>
<td>Writing of SQL queries based on parameters.</td>
<td>Section 5.2.1</td>
</tr>
<tr>
<td>2</td>
<td>Stored Procedures</td>
<td>Using stored procedures instead of SQL statements.</td>
<td>Section 5.2.2</td>
</tr>
<tr>
<td>3</td>
<td>Input Data Type Validation</td>
<td>The programmers must need to properly validate the input data types</td>
<td>Section 5.2.3</td>
</tr>
<tr>
<td>4</td>
<td>Input Variable Length Checking</td>
<td>Programmes need to validation checks in the program such as input variable length</td>
<td>Section 5.2.4</td>
</tr>
<tr>
<td>5</td>
<td>White List Filtering</td>
<td>Some of the special characters are normally used during injection attacks, so the developers should need to characterize such special characters as a blank list filtering.</td>
<td>Section 5.2.5</td>
</tr>
<tr>
<td>6</td>
<td>Smart Configuration Of DBMS</td>
<td>Defining different users with different privileges at DBMS level and using them in the development process</td>
<td>Section 5.2.6</td>
</tr>
<tr>
<td>7</td>
<td>Customized Error Message</td>
<td>Programmers can customize the error messages so that the attackers cannot get information regarding the database and other important objects.</td>
<td>Section 5.2.7</td>
</tr>
</tbody>
</table>

Table 5.2 Guidelines for Programmers
Chapter 6 – Results of the Guidelines Testing

6.1 Introduction

The steps described in the chapter 5 can be used as guidelines for programmers to protect the web applications from SQLi attacks in table 5.2. We identified the attacking types that are used by attackers to inject malicious code that is presented in chapter 4 using several different criteria. We first consider which attack type can be used by attackers, then we identified how to prevent SQLi cyber-attacks.

The literature study is used to find out the major attacks in web applications that is SQLi attack and the coding flaws behind those attacks. Then proof by demonstration way is used to test and evaluate some piece(s) of code that is described in the section 5.2. It can be used as the guidelines for the programmers. In this thesis, we gave guidelines to the programmers to prevent SQLi attack using literature study and proof by demonstration way.

6.2 Results of Guidelines with Respect to Attack Types

To evaluate our approach, we developed a web page prototype using C# and SQL Server with standard features, such as browsing products by name (see chapter 4 and 5). The code of the web page was intentionally written with weak or no input validation to ensure that web page is vulnerable to multiple types of SQLi attacks.
## Protecting Web Applications – Guidelines for the programmers.

### Table 6.1 Attacking Types and Guidelines

<table>
<thead>
<tr>
<th>Attacking Type</th>
<th>Name of Type</th>
<th>Description</th>
<th>Examples (if any)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tautology</td>
<td>This technique injects statements that are always true, so that the queries always return results upon evaluation of WHERE condition.</td>
<td>SELECT * FROM User_Details WHERE username = ‘*’ OR ‘1’ = ‘1’ AND password = ‘123’.</td>
<td>Section 4.2.1</td>
<td></td>
</tr>
<tr>
<td>Illego/Logically incorrect queries</td>
<td>The attacker injects malicious input in the query to produce some logical errors, in order to get the information.</td>
<td>xyz/profile.aspx?id=50’</td>
<td>Section 4.2.2</td>
<td></td>
</tr>
<tr>
<td>End of line comment</td>
<td>The attacker uses half detail and add -- in the end of the input field so that the rest of information is treated as a comment.</td>
<td>SELECT * FROM User_Details WHERE username = cde --</td>
<td>Section 4.2.3</td>
<td></td>
</tr>
<tr>
<td>Timing attack</td>
<td>The attacker collects information by observing the response time of the database.</td>
<td>SELECT * FROM User_Details WHERE id = ‘c12’ AND ascii(substring(password,1,1)) &gt; 71 waitfor delay ’0:0:7’ -- AND password = ’not required’;</td>
<td>Section 4.2.4</td>
<td></td>
</tr>
<tr>
<td>Union Query</td>
<td>The attacker injects query to concatenate with the original SQL query using the keyword UNION.</td>
<td>select username from User_Details where id = 5 union select ProcFrom tblProduct where id = 5;</td>
<td>Section 4.2.5</td>
<td></td>
</tr>
<tr>
<td>Blind SQL injection attacks</td>
<td>The attacker sends the database true or false questions and determines the answer based on the applications response in order to see if the db server responds in the same way as it does with a normal condition or different.</td>
<td></td>
<td>Section 4.2.6</td>
<td></td>
</tr>
<tr>
<td>Piggy backed queries/ Statement Injection</td>
<td>This is used when attacker knows little bit information related to database but not full details. Then attacker can use half detail and add -- in the end of the input field so that the rest of information is treated as a comment.</td>
<td>Pens’; delete * from tblProduct -</td>
<td>Section 4.2.7</td>
<td></td>
</tr>
</tbody>
</table>

### Instructions for Programmers

<table>
<thead>
<tr>
<th>Instructions for Programmers</th>
<th>Description</th>
<th>Examples (if any)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameterised Queries</td>
<td>Writing of SQL queries based on Parameters.</td>
<td>Select * from tblProduct where name = @name</td>
<td>Section 5.2.1</td>
</tr>
<tr>
<td>Stored Procedures</td>
<td>Using stored procedures instead of SQL statements.</td>
<td>create stored procedure and use that in coding part</td>
<td>Section 5.2.2</td>
</tr>
<tr>
<td>Input Data Type Validation</td>
<td>The programmers must need to properly validate the input data types.</td>
<td>In the case of numeric inputs, the programmer can simply reject any input that contains characters other than numbers.</td>
<td>Section 5.2.3</td>
</tr>
<tr>
<td>Input Variable Length Checking</td>
<td>Programmers needs to do validation checks in the program such as input variable length.</td>
<td></td>
<td>Section 5.2.4</td>
</tr>
<tr>
<td>White list filtering</td>
<td>Some of the special character are normally used during injection attacks, so the developer should need to characterize such special character as the black list filtering.</td>
<td>The filtering approach is suitable for the well structured data values, such as dates, email address etc.</td>
<td>Section 5.2.5</td>
</tr>
<tr>
<td>Smart Configuration of DBMS</td>
<td>Defining different users with different privileges at DBMS level and using them in the development process can be really helpful in mitigating the risk of SQL injection attack.</td>
<td>Define different users with different privilege</td>
<td>Section 5.2.6</td>
</tr>
<tr>
<td>Customized Error Messages</td>
<td>Programmers can customize the error messages so that the attackers cannot get information regarding the database and other important objects.</td>
<td></td>
<td>Section 5.2.7</td>
</tr>
</tbody>
</table>
Protecting Web Applications – Guidelines for the programmers.

We assumed that the programmers are able to correctly apply all required coding guidelines. In table 6.1, we evaluate the guidelines for programmers and in table 6.2, we summarize the results.

<table>
<thead>
<tr>
<th>Attacking type/Instructions</th>
<th>Parameterised Queries</th>
<th>Stored Procedures</th>
<th>Input Data Type Validation</th>
<th>Input Variable Length Checking</th>
<th>White list filtering</th>
<th>Smart Configuration of DBMS</th>
<th>Customized Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tautology</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Illegal/Logically incorrect queries</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>End of line comment</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Timing attack</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Union Query</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Blind SQLI attacks</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Piggy backed queries/Statement Injection</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 6.2 Consolidation Chart - Attacking Types versus Guidelines

From chapters 4 and 5, we have evaluated the final results in a tabular form, where we have described in short, the various attacking types used to attack web applications as mentioned in chapter 4 and the guidelines for the programmers that can help them to protect the web applications from these SQLi attacks. Table 6.1 shows attacking types and guidelines. Table 6.2 shows which attacks can be mitigated by which guidelines.
Chapter 7 - Conclusion and Future Work

7.1 Summary

There are many approaches to gain access to a database through a web application without permissions. Injection attack is the most common attacking way in these days and SQLi are the most frequently used approach.

Some organizations and companies may not know much about the SQLi attack. This thesis may be helpful to understand the SQLi attack, how the SQLi flow, recent attacks by SQLi, what are the coding flaws that can be exploited to attack vulnerable web applications and how programmers can mitigate these attacks.

The methodology applied for the research is literature study and we used the way proof by demonstration to get an overall picture and to contribute to the general awareness. The first step was to find out the coding flaws, then we designed guidelines that can help to protect web applications from SQLi attacks.

In our thesis we found that the practices such as parameterized statements in section 5.2.1 and stored procedures in section 5.2.2 should be implemented from the beginning.

For this we have first found the coding flaws using literature as mentioned in chapter 4 related to our RQ1. Then we have setup a programming environment using C# programming language and SQL Server. After this we have modified the flawed code part using parameterized queries in section 5.2.1 and stored procedures in section 5.2.2. According to the coding flaws in section 4, we mentioned the guidelines in section 5.2 that can help to protect our web applications from SQLi attacks which answers the RQ2. Afterwards, we have presented the guidelines with respect to attack type in a tabular form in table 6.1 and table 6.2.
7.2 Future Work

In future, the same work can be proceeded to check, if there are more coding flaws for SQLi attacks and how to avoid them so that our web applications can be safe. As we know, SQLi attack is not the only attack on web applications so in future, the work can also be extended to cover all the other types of attacks (refer to section 3.1), their flaws and the ways to mitigate them. Further, some software tool can be created in order to simulate the attack for research purpose and some software tools can also be developed to check, if the code is vulnerable or not.
References


Greenberg, A., (2013), Hacker group claims to have looted 100k via sql injection attack, http://www.scmagazine.com/hacker-group-claims-to-have-looted-100k-via-sql-injection-attack/article/317412/


Protecting Web Applications – Guidelines for the programmers.


Protecting Web Applications – Guidelines for the programmers.


