ADAMA SCIENCE & TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTING

DEPARTMENT OF COMPUTING

A MASTER'S THESIS

ON

Network Security Threat Vulnerability Prevention System for SQL Injection Attack

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN SOFTWARE ENGINEERING

By: TigistKorsa

January 2018

Adama, Ethiopia

ADAMA SCIENCE & TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTING

DEPARTMENT OF COMPUTING

Network Security Threat Vulnerability Prevention System for SQL Injection Attack

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN SOFTWARE ENGINEERING

By: TigistKorsa

Advisor: Dr. Satish Kumar (Ph.D.)

January 2018

Declaration

I declare that this study is my original work submitted in partial fulfillment of the requirement for the degree of Master of Science in software engineering and has not been submitted for any Degree or Diploma in any University or conference. To the best of my knowledge, all source of materials used for the study has been duly acknowledged. I have undertaken the study with the guidance and support of the research advisor.

TigistKorsa

Signature: _____

Advisor: Dr. Satish Kumar

Signature: _____

January 2018

ADAMA SCIENCE & TECHNOLOGY UNIVERSITY

SCHOOL OF GRADUATE STUDIES

DEPARTMENT OF COMPUTING

Vulnerability Prevention System for SQL Injection Attack Network Security Threat

Name and signature of members of the examining board

Name	**Signature**	**Date**
_____	_____	_____
Advisor		
_____	_____	_____
External Examiner		
_____	_____	_____
Internal Examiner		

Acknowledgements

First and foremost extraordinary thanks go for my Almighty God. It is with immense gratitude that I acknowledge Dr.satish Kumarmy thesis supervisor for providing me with many valuable ideas and insights. The successful completion of my thesis work is all due to his guidance and motivation. My sincere thanks also goes to Dr. MesfineAbebefor his guidance helped me in all the time of research and writing of this thesis. It was an enormous pleasure for me to work under his supervision. I am highly grateful to my parents; they taught me the right, encouraged me and gave me hope and unconditional love. I wish to both of them happiness and well health. My friends,words are few to express my deepest thanks to you. You were always supporting me and encouraging me with your best wishes. The last but not the least thanks goes to my husband, you are the motivating force behind me at all times through both ups and downs of my educational life. Thank you for everything you give me.

Abstract

Over the past decade, the web has been embraced by millions of businesses as an inexpensive channel to communicate and exchange information with prospects and transactions with customers. Web applications are computer programs allowing website visitors to submit and retrieve data to or from a database over the Internet using their preferred web browser. The data is then presented to the user within their browser as information is generated dynamically (in a specific format, e.g. in HTML using CSS) by the web application through a web server. Due to feature of web application, it is possible to perform various attacks against web applications. SQL injection is one of the oldest, most dangerous of web application vulnerabilities. Attackers can execute malicious SQL command that allow them to control a web application database. It lets attackers access or delete data and change application database. SQL injection vulnerability could affect any website or web application and it occurs when an application uses untrusted data. SQL injection, also known as SQLIA, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details. SQL injection can have the impact on a business.A successful attack may result in the unauthorized viewing user lists, the deletion of entire tables and, in certain cases, the attacker gaining administrative rights to a database, all of which are highly detrimental to a business. In this thesis we have proposed filter mechanism to prevent SQL Injection in a web application and we designed SQLIA signature pattern Filter which we have placed in between the user and the application server to intercept the entire request coming from the user. The filter program analyzes the input data to detect attack patterns. If any pattern matches with the attack signature then it redirects the request to the page which display error

message. Our filter approach has been implemented successfully and fully able to fix SQLIA vulnerabilities. The evaluation of performance and implementation of the proposed SQLIA prevention system are made with PHP server scripting language. The results obtained by the implementation and evaluation are measured in number of test cases. The result shows that the output is encouraging and further refinement of the work can produce more robust and reliable SQLIA prevention system.

List of Abbreviation

AES	Advanced Encryptionstandard
ANSI	American National Standards Institute
ASP	Active Server Page
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheet
DOS	Denial OfService
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
JSP	Java Server Page
OWASP	Open Web Application Security Project
PHP	Hypertext Preprocessor
RDBMS	Relational Database Management System
RFI	Remote File Inclusion
RMI	Remote Method Invocation
SQL	Structured Query Language
SQLIA	Structured Query Language Injection Attack
URL	Uniform Resource Locator
VS	Vulnerability Scanner
WAMP	Windows ApacheMYSQL And PHP.

WEBSSARI Web Application Security ByStaticAnalysis And RuntimeInspection

XHTML Extensible Hypertext Markup Language

XML Extensible Markup Language

XSS Cross-Site Scripting

List of Figures

List of Tables page

Table Contents

CHAPTER ONE: INTRODUCTION

In this chapter the background of the research and the rationale initiated to do this research has been presented. It presents the overall objective, methodology, significance of the study, scope, and limitation of the thesis and problem that has to be addressed finally the organization of the thesis is presented.

1.1.Background of the study

According to KASPERSKY Security Bulletin overall statistics, in 2015, there were 1,966,324 registered notifications about attempted malware infections that aimed to steal money via online access to bank accounts. Ransomware programs were detected on 753,684 computers of unique users; 179,209 computers were targeted by encryption ransomware. KASPERSKY Lab's web antivirus detected 121,262,075 unique malicious objects: scripts, exploits, executable files, etc. KASPERSKY Lab solutions repelled 798,113,087 attacks launched from online resources located all over the world. 34.2% of user computers were subjected to at least one web attack over the year. To carry out their attacks, cybercriminals used 6,563,145 unique hosts. 24% of web attacks neutralized by KASPERSKY Lab products were carried out using malicious web resources located in the US. KASPERSKY Lab's antivirus solutions detected a total of 4,000,000 unique malicious and potentially unwanted objects. [1].90% of large organizations reported that they had suffered a security breach, up from 81% in 2014."50% of small organizations spent up to £999 cash to recover from their worst security incident of the year" [2] Computer threats can affect a system's operation or cause disastrous and extensive data corruption. Prevention is better than cure, so it is important to detect vulnerabilities. Preventing vulnerabilities to threats does not require much time and effort than correcting. It starts by simply opening an unknown, suspicious or untrustworthy attachment to an email, using illegal software, using weak passwords, sending unencrypted data, downloading unknown content from unknown websites are very few examples of exposure to threats.When many different kinds of information technology tools or devices come together and connected to perform a wonderful job, the threat escalates much harder. In recent years, attacks are having more complex forms in becoming

well-organized, well-financed, technically advanced, hidden long term presence and even undetectable. [3] Whatsoever, we need computers in our daily life to get better results even under worse environment. New web-based attack types and vectors are coming out every day. This is causing businesses, communities and individuals to take security seriously now more than they ever have in the past. This is a huge win for the World Wide Web and it's a trend that is pushing technology further towards more robust and securely developed web applications.

The motive behind online attacks is varied. organization had stuff to steal, their site could be used to display propaganda or broadcast spam, or maybe they just forgot to update and organization forgetfulness was able to satisfy the bored desires of a hackers. One of those reasons is why websites got hacked. Every site can serve a purpose: to hold sensitive data, or at the very least, provide usable resources to send spam or attack other targets. When it comes to more complex cases it includes the following.

· **Cross Site Scripting (XSS)** XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc. Often misunderstood, and even more often underestimated, XSS is a style of attack where the front of the website acts as a launching point for attacks on other users visiting the website. This happens when developers don't properly test their code for the possibility of allowing scripts to be injected. The scripts can then be executed without the site's original functionality intending them to be. If XSS vulnerability is present on a website, then an attacker can craft code that executes when other users open the same website. This causes the new users to interact with the malicious background entity created by the attacker. Once a connection has been initiated, usually via social-engineering tactics convincing a user to do something they shouldn't, the attacker is able to infiltrate your website visitors' computers. [5,6]

· **Injection Flaws** Injection flaws, particularly SQL injection, are common in web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data. Injection vulnerabilities are rated as the number one problem on the list of top 10 security issues put out by Open Web Application Security Project and continue to be a major source of concern for application and web developers looking to utilize the benefits of

storing usable information in a local database. [5] Due to the predictable nature of these types of applications, an attacker can craft a string using specific Structured Query Language (SQL) commands, and know it can be used to force the database to give up the goods. These strings can be entered in places like search boxes, login forms, and even directly into a URL to negate simple client-side security measures on the page itself. Why is this so dangerous? The database keeps the most important and lucrative space on a system, and can not only be coaxed to give up usernames, passwords, and credit card numbers, but can also be attacked in a way that can give an attacker a foothold to gain access to the entire system, and to every other database instance housed there for other websites and applications.[6, 7]

· **Malicious File Execution** Code vulnerable to remote file inclusion (RFI) allows attackers to include hostile code and data, resulting in devastating attacks, such as total server compromise. Malicious file execution attacks affect PHP, XML and any framework which accepts filenames or files from users.

· **Insecure Direct Object Reference** A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization.

· **Cross Site Request Forgery (CSRF)** A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.

· **Information Leakage and Improper Error Handling** Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data, or conduct more serious attacks.

· **Broken Authentication and Session Management** Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities.

· **Insecure Cryptographic Storage** Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes, such as credit card fraud.

· **Insecure Communications** Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications.

· **Failure to Restrict URL Access** Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly.

One of the techniques available to the would-be information thieves is SQL-Injection (SQL-I). SQLI attacks involve a variety of methods, but the intention of an attacker using it is to submit specially-chosen patterns when asked for the user's username and password on an internet form. The values passed from a form may be directly concatenated into a string with an SQL query string. Then the resulting SQL query string is dynamically parsed in order to test.

For example, if the username and password are correct. When the text from the user input is unchecked and incorporated into the SQL query, these abnormal values can result in highly abnormal behavior such as gaining access to the system despite not supplying the correct password, selecting columns from completely different tables than the ones being queried in the original query, or even highly dangerous behavior such as deleting any tables. It is therefore imperative that these values be checked before being submitted to a database management system (DBMS) parser to be run on the database.

1.2.Statement of the Problem

According to OWASP ans SANS report out of Ten Most Critical Web Application Security Risks SQL-injections was ranked number one of the most dangerous from 2013-2017, and it is still in the top of critical security risks (Top 10 2017, May-13). SQL-injections are not only one of the most critical attacks, it is also one of the most popular data extraction techniques (X-force trend and risk report, May-13). In January to June 2011 SQL-injections accounted for 68 percent of the total number of web-application attacks [8].Every organization should do as much as possible with full effort in order to avoid SQLI vulnerabilities.As devices get connected more and more, the risk of vulnerability for threats grows proportionally. The obvious quote that applies here is "Prevention is better than cure". As a latecomer advantage, the developing countries have the opportunity to learn from the success and failure of the developed countries which will be taken as an input in this research. Very mature security standards and models will be used as a base for evaluating the status of vulnerability of

organizations. As a developing country, we suffer from the security challenges imposed by globalization. The developed countries are more experienced and they can handle threats at almost any expense. We as developing countries have to focus on prevention of threats and especially of Vulnerabilities to threats. We cannot get away from information technology and live in an island of non-globalization.

This thesis investigating the problem domain, provide solution and will also contribute to the general awareness of SQL-injections.In addition the research needs to answer the following questions.

- ✓ What are currently and widely used SQL-I techniques?
- ✓ How can we minimize SQL-I vulnerabilities?
- ✓ How SQLI attack Prevention is better than cure?
- ✓ Why SQL-I attack vulnerability prevention systems are needed in any networked organization?

1.3.Motivation

As protection of the network threat there are lots of tools to detect the vulnerability of the computer network. But still lots of organization use network technology are vulnerable for network threat.special organization connect to internet and have web application more vulnerable for network threat. Web applications including some of our government institution and private institutions websites are vulnerable for SQLI.The reasons are the security misconception and hole of the network.So it is urgent to study how network vulnerability occurs and what effective countermeasure can prevent the network threat .So these issues inspire me to study security solutions which do not need any extra overhead.

1.4. Objective of the research

General objective

The general objective of this research is to study SQL-I attack vulnerability protection security solutions which do not need any extra overhead, as well as a security mechanism which can be embedded with the web application.

Specific objectives

- ✓ To conduct a detail literature review to understand the SQL-I attack.
- ✓ To Design system architecture.
- ✓ To develop filter algorithm and block algorithm
- ✓ To provide a SQL-I attack vulnerability prevention system.
- ✓ To evaluate the performance of proposed prevention system

1.5. Scope of the study

This study Provide a SQL-I attack vulnerability prevention system .It focuses on identifying possible vulnerability coding practices and prevention for vulnerability. However, this work does not focus on SQLI vulnerability detection system that can take an action for the coming intrusion.

1.6. Significant of the study

A web-based application is an application that uses a website as the interface or front-end. Users can easily access the application from any computer connected to the Internet using a standard browser. Web-based applications are easier to develop, more useful for your users, easier to install, maintain and keep secure and easier to grow as organizations grow. When it comes to attacking web applications, the attacker will try several means of compromising the application. But there is one that is particularly prized where a backend database is used: executing arbitrary SQL commands. This research significant of this research list below

- ✓ Minimizes the cost of huge budget needed to recover from information network attack

- ✓ Avoids attacks prior happening
- ✓ Maintains a stable working environment

1.7. Research methodology

Different methodologies will be employed in this research in order to accomplish the general and specific objectives of this study. The first thing to do is to conduct a comprehensive review of literatures to acquire a deeper understanding of the research area and its problem domains. The other thing to do is system architecture and Implementation s and Evaluation.

1.8. Organization of the paper

The rest of the study is organized as follows: Chapter two covered related literatures on the basic concepts of architecture of web application and how it relate to SQL injection, overview of SQL injection: attack intent ,methods of SQL injection ,injection mechanism, type of SQL injection, SQL injection discovery techniques ,impact of SQL and automated tools for SQL injection attack detection Chapter three describes the research methodology and strategy that aims to identify the potential SQL injection prevention solutions that could be applied to web application. It begins by describing exploratory research approaches. Primary data collection and analysis technique were described as information acquisition method. Validity and reliability requirements also identified and presents the results of the interview, techniques and technologies required to design the prototype. In Chapter five, backgrounds of a framework for building the proposed SQL injection prevention system for web security threats, the detail elements of the proposed SQL injection prevention system, what design is better for SQLIA, why the specific design was selected for SQLIA also discussed. In chapter five theimplementationof the SQLIAprevention system have discussed. Finally, in chapter six conclusions about the research and suggestions for future research direction were presented

CHAPTER TWO: LITERATURE REVIEW

2.1. Overview

This Chapter is the output of literature survey of web server vulnerability for SQL injection attacks. Vulnerability is a weakness which allows an attacker to reduce a system's information assurance. It is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw. "Vulnerability constitutes any known weakness on a system that could potentially be exploited by malicious software or hacker." [12]. the vulnerability should be weighed against what kind of business it is, a bank or perhaps a government institution. Security vulnerabilities associated with computer networks have risen among the foremost concerns for network and security professionals because it consistently provides serious threats to the efficiency and effectiveness of organizations [13]. Mainly there are two categories to SQL injection attack that is Normal injection and Blind injection. This Chapter discusses about overview of SQL injection, Type of SQL injection attacks, SQL injection

2.2. Web applications overview

The web is a highly programmable environment that allows mass customization through the immediate deployment of a large and diverse range of applications, to millions of global users. Two important components of a modern website are flexible web browsers and web applications; both available to all and sundry at no expense. Web browsers are software applications that allow users to retrieve data and interact with content located on web pages within a website. Today's websites are a far cry from the static text and graphics showcases of the early and mid-nineties: modern web pages allow personalized dynamic content to be pulled down by users according to individual preferences and settings. Furthermore, web pages may also run client-side scripts that "change" the Internet browser into an interface for such applications as web mail and interactive mapping software (e.g., Yahoo Mail and Google Maps).Most importantly, modern web sites allow the capture, processing, storage and transmission of sensitive customer data (e.g., personal details, credit card numbers, social security information, etc.) for immediate and recurrent use. And, this is done through web applications. Such features as webmail, login pages, support and

product request forms, shopping carts and content management systems, shape modern websites and provide businesses with the means necessary to communicate with prospects and customers. These are all common examples of web applications. Web applications are, therefore, computer programs allowing website visitors to submit and retrieve data to/from a database over the Internet using their preferred web browser. The data is then presented to the user within their browser as information is generated dynamically (in a specific format, e.g. in HTML using CSS) by the web application through a web server. The web browser is key – it interprets and runs all scripts etc. while displaying the requested pages and content. Wikipedia brilliantly terms the web browser as the "universal client for any web application". Another significant advantage of building and maintaining web applications is that they perform their function irrespective of the operating system and browsers running client side. Web applications are quickly deployed anywhere at no cost and without any installation requirements at the user's end. As the number of businesses embracing the benefits of doing business over the web increases, so will the use of web applications and other related technologies continue to grow.

2.2.1. Three-layered web application model

The figure 2.1 details the three-layered web application model. The first layer is normally a web browser or the user interface; the second layer is the dynamic content generation technology tool such as Java servlets (JSP) or Active Server Pages (ASP), and the third layer is the database containing content (e.g., news) and customer data (e.g., usernames and passwords, social security numbers and credit card details).

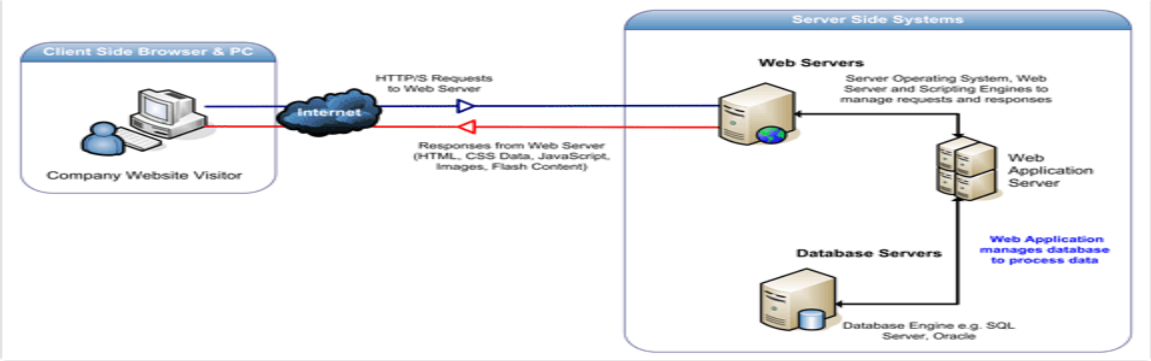

Figure 2.1: Details the three-layered web application model

2.3. Web application attack

Web applications do raise a number of security concerns stemming from improper coding. Serious weaknesses or vulnerabilities allow hackers to gain direct and public access to databases in order to churn sensitive data. This is known as a web application attack

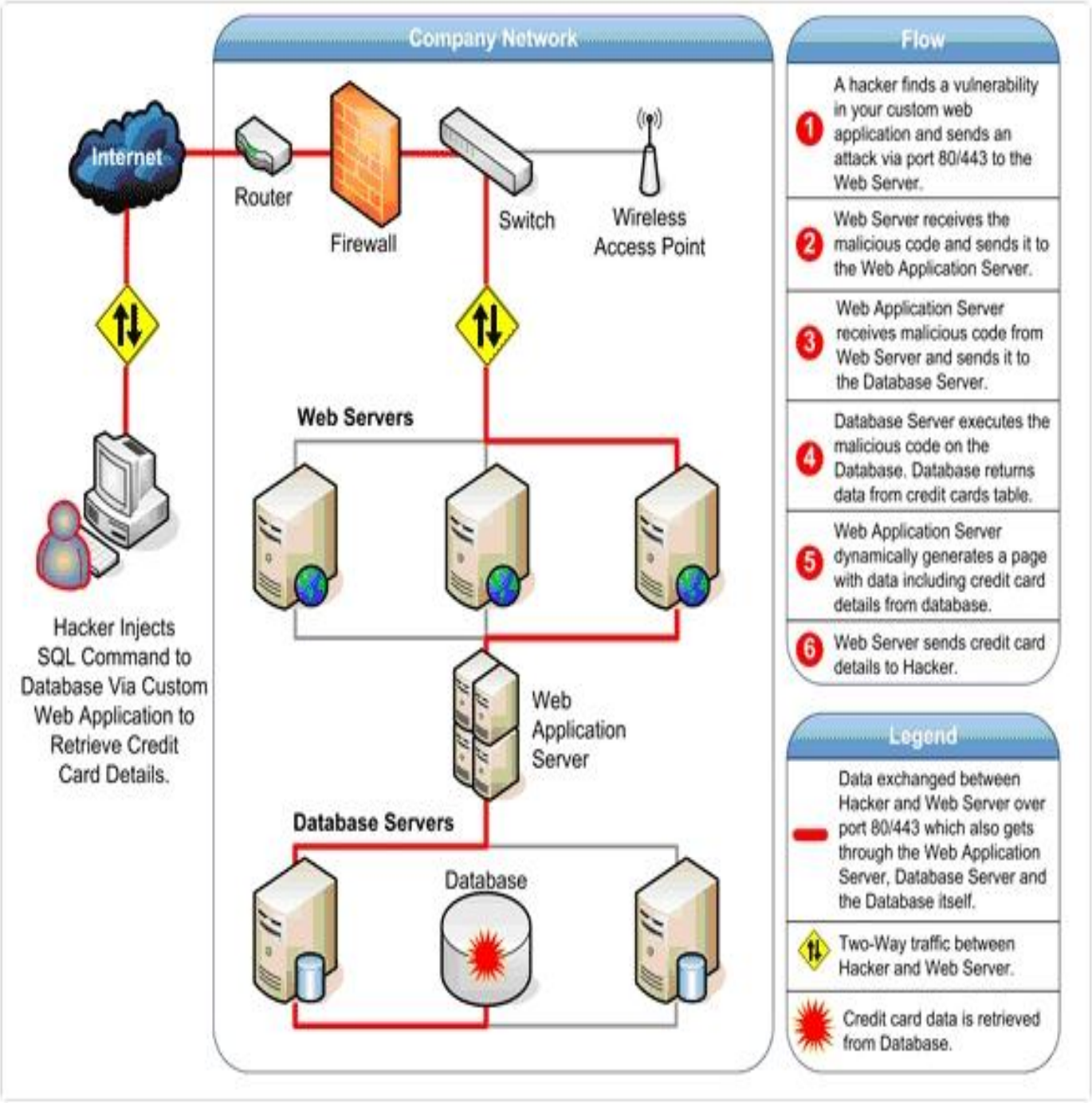

Figure 2.2 how a hacker can access the data residing on the database

Many of the databases contain valuable information (e.g., personal and financial details) making them a frequent target of hackers. Although such acts of vandalism as defacing corporate websites are still commonplace, nowadays, hackers prefer gaining access to the sensitive data residing on the database server because of the immense pay-offs in selling the data

A hacker can quickly access the data residing on the database through a dose of creativity and, with luck, negligence or human error, leading to vulnerabilities in the web applications (figure 2.2).As stated, websites depend on databases to deliver the required information to visitors. If web applications are not secure, i.e., vulnerable to, at least one of the various forms of hacking techniques, then your entire database of sensitive information is at serious risk of a web application attack. Recent research shows that 75% of cyber-attacks are done at web application level.[14]

Websites and related web applications must be available 24 hours a day, 7 days a week, to provide the required service to customers, employees, suppliers and other stakeholders. Firewalls and SSL provide no protection against a web application attack, simply because access to the website has to be made public. All modern database systems (e.g. Microsoft SQL Server, Oracle and MYSQL) may be accessed through specific ports (e.g., port 80 and 443) and anyone can attempt direct connections to the databases effectively bypassing the security mechanisms used by the operating system. These ports remain open to allow communication with legitimate traffic and therefore constitute a major vulnerability. Web applications often have direct access to backend data such as customer databases and, hence, control valuable data and are much more difficult to secure. Those that do not have access will have some form of script that allows data capture and transmission. If a hacker becomes aware of weaknesses in such a script, he may easily reroute unwitting traffic to another location and illegitimately hive off personal details. Most web applications are custom-made and, therefore, involve a lesser degree of testing than off-the-shelf software. Consequently, custom applications are more susceptible to attack, Web applications, therefore, are a gateway to databases especially custom applications which are not developed with security best practices and which do not undergo regular security audits .One of the oldest, most dangerous of web application vulnerabilities SQL injection. Attackers can execute malicious SQL command that allow attackers to control a web application database. [13]

2.4.History and overview of SQL injection Attack

A famous hacker known as Rain Forrest Puppy is usually credited as the inventor of SQL injection. SQL injection is an attack technique used by hackers to exploit web sites by altering

backend SQL statements through manipulating application input. SQL Injection happens when a developer accepts user input that is directly placed into a SQL Statement and doesn't properly filter out dangerous characters, as shown Figure 2.3 This can allow an attacker to not only steal data from the database, but also modify and delete it. Certain SQL Servers such as Microsoft SQL Server contain Stored and Extended Procedures (database server functions).

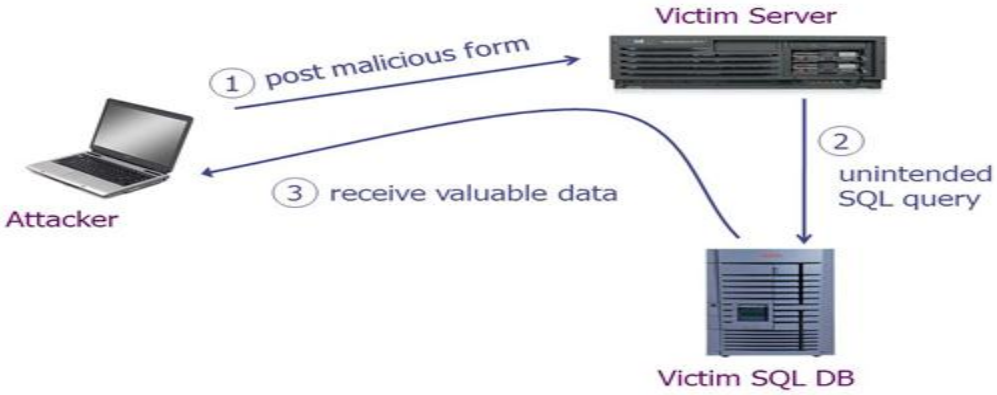

Figure 2.3: Basic picture of SQL Injection[48]

If an attacker can obtain access to these Procedures it may be possible to compromise the entire machine. Attackers commonly insert single quotes into a URL's query string, or into a forms input field to test for SQL Injection.The intention of SQLIA are Adding or modifying data, Extracting data, Performing denial of service, bypassing authentication and Executing remote commands .The attacker use different method to attack the web server like Injection through user input, malicious strings in web forms, Injection through cookies, Modified cookie fields contain attack strings, Injection through server variables and Headers are manipulated to contain attack strings. SQL-injections are not only one of the most critical attacks; it is also one of the most popular data extraction techniques.[15]. In January to June 2011 SQL-injections accounted for 68 percent of the total number of web-application attacks (see Figure 2.4) [8].

Total web application attacks January-June 2011

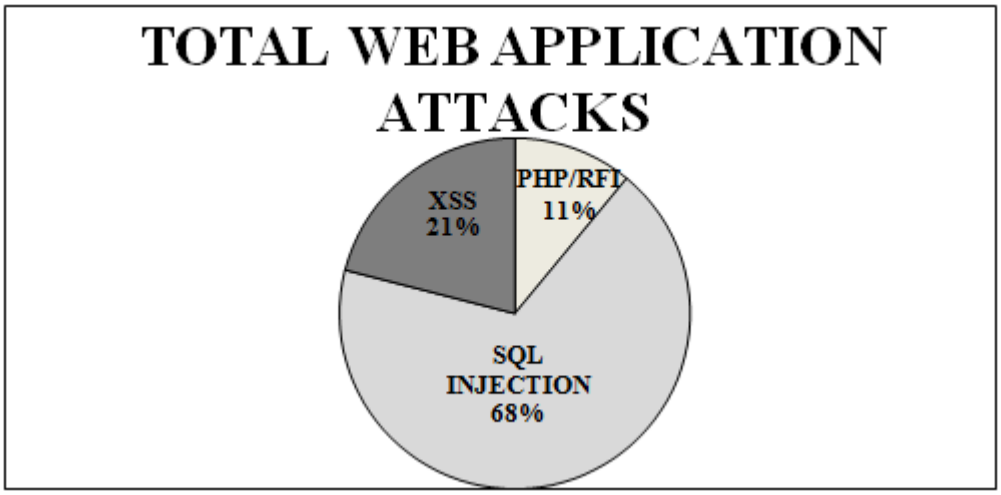

Figure 2.4: reported attacks against website [15]

Some major hacks regarding SQL injection listed in "History of SQL Injection"[Online].Available:http://hackertarget.com/10-years-of-SQL-injection:[16]
July 2012, Yahoo confirms 4 million accounts hacked, June 2011Hactivist 'Lulzsec' breached website of SONY, May 2011 COMODO Brazil breached, March 2011 Official homepage of MYSQL website compromised, November 2010 Royal Navy website attacked, January 2009 Heartland payment systems breached, June 2007 and Microsoft UK website defaced.
In the following subchapters the most common SQL-I vulnerabilities in web server and their type are described.

2.5.SQL injection Attack Intent

Attacks can also be characterized based on the goal or intent of the attacker.[24] Therefore, each of the attack type definitions that we explain in Section 2.8 includes a list of one or more of the attack intents defined in this section.
Identifying injectable parameters: The attacker wants to probe a Web application to discover which parameters and user-input fields are vulnerable to SQLIA.
Performing database fingerprinting: The attacker wants to discover the type and version of database that a Web application is using. Certain types of databases respond differently to different queries and attacks, and this information can be used to "fingerprint" the database.

Knowing the type and version of the database used by a Web application allows an attacker to craft database specific attacks.

Determining database schema: To correctly extract data from a database, the attacker often needs to know database schema information, such as table names, column names, and column data types. Attacks with this intent are created to collect or infer this kind of information.

Extracting data: These types of attacks employ techniques that will extract data values from the database. Depending on the type of the Web application, this information could be sensitive and highly desirable to the attacker. Attacks with this intent are the most common type of SQLIA.

Adding or modifying data: The goal of these attacks is to add or change information in a database.

Performing denial of service: These attacks are performed to shutdown the database of a Web application, thus denying service to other users. Attacks involving locking or dropping database tables also fall under this category.

Evading detection: This category refers to certain attack techniques that are employed to avoid auditing and detection by system protection mechanisms.

Bypassing authentication: The goal of these types of attacks is to allow the attacker to bypass database and application authentication mechanisms. Bypassing such mechanisms could allow the attacker to assume the rights and privileges associated with another application user.

Executing remote commands: These types of attacks attempt to execute arbitrary commands on the database. These commands can be stored procedures or functions available to database users.

Performing privilege escalation: These attacks take advantage of implementation errors or logical flaws in the database in order to escalate the privileges of the attacker. As opposed to bypassing authentication attacks, these attacks focus on exploiting the database user privileges.

2.6.Methods of SQL injection

Error Based SQL Injection

In this type of attacks, when an attacker tries to execute SQL query, the server returns an error page to the user which describes the type and cause of the error in detail. Thus, the attacker can try to match his query with the developers query by using the information contained in the error messages returned in response by the database server. It consist two subtypes of injection: Union Based and Error Based.

Blind SQL Injection

Blind SQL injection is similar to normal SQL Injection except that when an attacker attempts to exploit an application, instead of getting a useful error message, he gets a generic page specified by the developer. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An unsuccessful injection might redirect the attacker to the standard error page. A successful injection might display a blank page. Usually, it's a matter of patience and skill before an attacker can successfully blind inject a website [18].

2.7. Injection Mechanisms

Malicious SQL statements can be introduced into a vulnerable application using many different input mechanisms. In this section, we explain the most common mechanisms.

Injection through user input In this case, attackers inject SQL commands by providing suitably crafted user input. A Web application can read user input in several ways based on the environment in which the application is deployed. In most SQLIAs that target Web applications, user input typically comes from form submissions that are sent to the Web application via HTTP GET or POST requests [14]. Web applications are generally able to access the user input contained in these requests as they would access any other variable in the environment.

Injection through cookies: Cookies are files that contain state information generated by Web applications and stored on the client machine. When a client returns to a Web application, cookies can be used to restore the client's state information. Since the client has control over the storage of the cookie, a malicious client could tamper with the cookie's contents. If a Web application uses the cookie's contents to build SQL queries, an attacker could easily submit an attack by embedding it in the cookie [23]. Unlike other parameters, cookies are not supposed to be handled by users. Outside of session cookies which are random, cookies may contain data in clear or encoded in hexadecimal, base64, hashes (MD5, SHA1), and serialized information. If we can determine the encoding used, we can attempt to inject SQL commands.

For example:

```
functionis_user($user) {
```

```
global $prefix, $db, $user_prefix;

if(!is_array($user)) {

$user = base64_decode($user);

$user = explode(“:”, $user);

$uid = “$user[0]”;

$pwd = “$user[2]”;

} else {

$uid = “$user[0]”;

$pwd = “$user[2]”;}

if ($uid != “” AND $pwd != “”) {

$sql = “SELECT user_password FROM “.$user_prefix.”_users WHERE user_id=’$uid'”;

$result = $db->sql_query($sql);

$row = $db->sql_fetchrow($result);

$pass = $row[user_password];

if($pass == $pwd&& $pass != “”) {

return 1;}}return 0;}
```

The cookie contains base64 encoded form identifier, a field that is unknown and a password. If we use as a cookie 5678 ‘ORG SELECT’ mypass ‘:: passbook base64 encoded, the SQL query becomes:

SELECT user_password FROM nk_users WHERE user_id=’5678′ ORG SELECT ‘passbook’

This query returns the password passbook, the same password as we have to provide. So we are connected.There are many HTTP interceptors and HTTP editors that can intercept the HTTP request before it is sent to the server. Then the tester can introduce his malicious SQL statement in the cookie field.

Injection through server variables: Server variables are a collection of variables that contain HTTP, network headers, and environmental variables. Web applications use these server variables in a variety of ways, such as logging usage statistics and identifying browsing trends. If these variables are logged to a database without sanitization, this could create SQL injection vulnerability [18,19]. Because attackers can forge the values that are placed in HTTP and network headers, they can exploit this vulnerability by placing an SQLIA directly into the headers. When the query to log the server variable is issued to the database, the attack in the forged header is then triggered.

Second-order injection: In second-order injections, attackers seed malicious inputs into a system or database to indirectly trigger in SQLIA when that input is used at a later time. The objective of this kind of attack differs significantly from a regular (i.e., first-order) injection attack. Second-order injections are not trying to cause the attack to occur when the malicious input initially reaches the database. Instead, attackers rely on knowledge of where the input will be subsequently used and craft their attack so that it occurs during that usage.

To clarify, in the example, a user registers on a website using a seeded user name, such as "admin' --". The application properly escapes the single quote in the input before storing it in the database, preventing its potentially malicious effect.The user modifies his or her password, an operation that typically involves (1) checking that the user knows the current password and (2) changing the pass-word if the check is successful. To do this, the Web application might construct an SQL command as follows:

queryString="UPDATE users SET password='" + newPassword +"' WHERE userName='" + userName + "' AND password='" +oldPassword + "'"

newPassword and oldPassword are the new and old passwords, respectively, and username is the name of the user currently logged-in (i.e.,''admin'--''). Therefore, the query string that is sent to the database is (assume that newPassword and oldPassword are "newpwd" and"oldpwd"):

UPDATE users SET password=’newpwd’ WHERE userName= ’admin’--’ AND password=’oldpwd’

Because “--” is the SQL comment operator, everything after it is ignored by the database. Therefore, the result of this query is that the database changes the password of the administrator (“admin”) to an attacker-specified value. Second-order injections can be especially difficult to detect and prevent because the point of injection is different from the point where the attack actually manifests itself. A developer may properly escape, type-check, and filter input that comes from the user and assume it is safe. Later on, when that data is used in a different context, or to build a different type of query, the previously sanitized input may result in an injection attack.

2.8.SQL Injection Discovery Technique

It is not compulsory for an attacker to visit the web pages using a browser to find if SQL injection is possible on the site. Generally attackers build a web crawler to collect all URLs available on each and every web page of the site. Web crawler is also used to insert illegal characters into the query string of a URL and check for any error result sent by the server. If the server sends any error message as a result, it is a strong positive indication that the illegal special Meta character will pass as a part of the SQL query, and hence the site is open to SQL Injection attack.[20, 21]

2.9.Impact of SQL injection attack on web server

An attacker sends intentionally malformed requests to a company’s website hoping that the server will malfunction and either return non-public data in response to the request or grant the attacker deep, administrative access to the server. The risk factors associated with SQL injection are as below; the platform affected with SQLIA can be as:

- ✓ Language: SQL
- ✓ Platform: Any (requires interaction with a SQL database)

SQL injection has become common issue with database driven web sites and web applications. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind. Essentially

the attack is accomplished by placing a Meta character into data input to then place SQL commands in the control plane, which did not exist there before. This flaw depends on the fact that SQL makes no real distinction between the control and data planes. The main impact of SQL Injection attacks is as below:

Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerability

Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to Connect to a system as another user with no previous knowledge of the password. [13]

Authorization: If authorization information is held in a SQL database, it is very much possible to change this information through the successful exploitation of SQL Injection vulnerability.

Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection.

2.10. Type of SQL injection Attacks

In this section, we present and discuss different kinds of SQLI attacks known to date, as shown figure 2.3. For each attack type, we deliberate a description of the attack, one or more attack intents, an attack example, and a set of references to publications and Websites that discuss the attack technique and its variations in greater detail. The different types of attacks are generally not performed in isolation; many of them are used together or sequentially, depending on the specific goals of the attacker.[22,24].

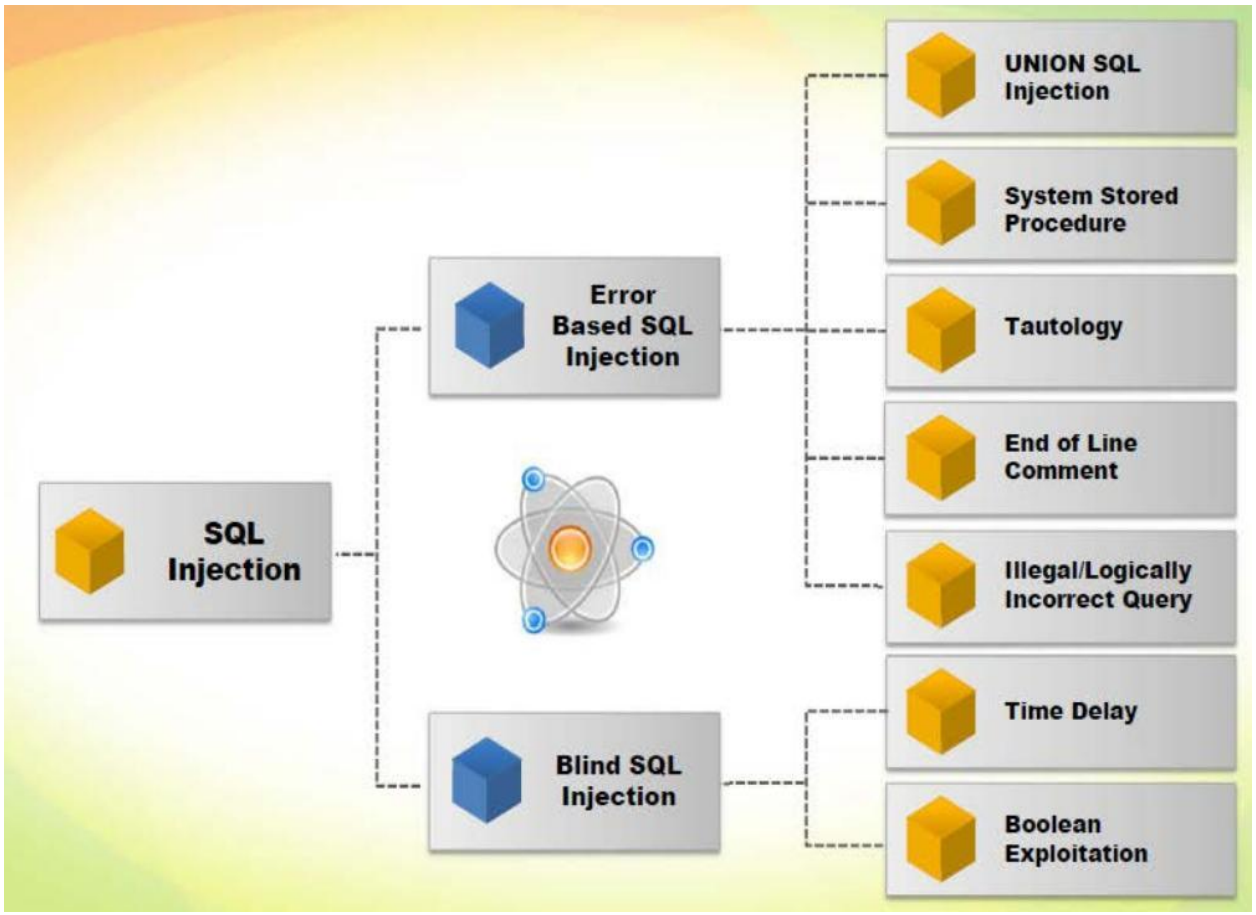

Figure 2.5.Type of SQL injection

I. Tautologies

Attack Intent: Bypassing authentication; identifying inject able parameters; extracting data.

Tautologies attack is the simplest and top known SQL injection attack. This type of attack inserts malicious code into SQL queries and modifies the conditional statement. The most significant characters of this type of attack are **three key components**: **making one condition always true; OR; comment mark to other condition**. The condition statement of SQL query restricts that retrieved only fulfills the specific condition. The retrieved data will be displayed all row data in one table of database if the condition statement is deleted by tautologies attack (liu&xu 2013). The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The most common usages are to bypass

authentication pages and extract data. In this type of injection, an attacker exploits an injectable field that is used in a query's WHERE conditional. Transforming the conditional into a tautology causes all of the rows in the database table targeted by the query to be returned. In general, for a tautology-based attack to work, an attacker must consider not only the inject able/vulnerable parameters, but also the coding constructs that evaluate the query results. (Halfond, Viegas, & Alessandro, 2006)

Example: an attacker submits " ' or 1=1 - -" for the *login* input field

(The input submitted for the other fields is irrelevant).

The resulting query is:

SELECT accounts FROM users WHERE

login=' ' or 1=1 -- AND pass='' AND pin=

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application. Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of them.

In the example, the returned set evaluates to a non-null value, which causes the application to conclude that the user authentication was successful. Therefore, the application would invoke method display Accounts () and show all of the accounts in the set returned by the database.

II. Illegal/Logically Incorrect Queries

Attack Intent: Identifying inject able parameters, performing database fingerprinting and extracting data.This attack lets the attacker gather important information about the type and structure of the back-end database of an application. The attack is considered a preliminary, information gathering step for other attacks. The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive; originally intended to help programmers debug their applications, further helps attackers gain information about the schema of the back-end database. When performing this attack, an attacker tries to inject statements that cause a syntax, type conversion, or logical error into the database. Syntax errors can be used to identify inject able parameters. Type errors can be used to deduce the data types of certain columns or to extract data. Logical errors often reveal the names of the tables and columns that caused the error. The specific attacks within this class are largely the same as those used in a Tautological attack. The difference is that these are intended to determine how the

system responds to different attacks by looking at the response to a normal input, an input with a logically true statement appended (typical tautological attack), an input with a logically false statement appended (to catch the response to failure) and an invalid statement to see how the system responds to bad SQL. This will often allow the attacker to see if an attack got through to the database even if the application does not allow the output from that statement to be displayedIn fact, the attacker may initially use a bot to detect a vulnerable web site and then recursively use this class of attack forensically to learn application and database specifics.

Example: This example attack's goal is to cause a type conversion error that can reveal relevant data.

To do this, the attacker injects the following text into input field pin: "convert (int,(select top 1 name from sysobjects where xtype='u'))".

The resulting query is:

SELECT accounts FROM users WHERE login='' AND pass='' AND pin= convert(int,(select top 1 name from sysobjects where xtype='u'))

In the attack string, the injected select query attempts to extract the first user table (xtype='u') from the database's metadata table (assume the application is using Microsoft SQL Server, for which the metadata table is called sysobjects). The query then tries to convert this table name into an integer. Because this is not a legal type conversion, the database throws an error.

For Microsoft SQL Server, the error would be: *"Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value 'CreditCards' to a column of data type int."*

There are two useful pieces of information in this message that aid an attacker. First, the attacker can see that the database is an SQL Server database, as the error message explicitly states this fact. Second, the error message reveals the value of the string that caused the type conversion to occur. In this case, this value is also the name of the first user-defined table in the database: "CreditCards." A similar strategy can be used to systematically extract the name and type of each column in the database. Using this information about the schema of the database, an attacker can then create further attacks that target specific pieces of information.

An attacker gathers important information about the type and structure of the back-end database of a Web application by injecting illegal or logically incorrect SQL syntax which will make the application return default error pages that often reveal vulnerable/injectable parameters to the

attacker. This attack is considered as a preliminary, information-gathering step for other SQL injection attacks.

For example:

Searching column name

Input (username): *'ddd"*

SQL: *SELECT * FROM students WHERE username = 'ddd"' AND password =*

Result:*"Incorrect syntax near 'ddd'. Unclosed quotation mark after the character string '' AND Password='aaa"."*

III. Union Query

Attack Intent: Bypassing Authentication; extracting data. In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different than the one that was intended by the developer. The attacker can insert any appropriate query to retrieve information from a table different from the one that was the target of the original statement. Attackers do this by injecting a statement of the form:

UNION SELECT <rest of injected query>.

Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. The database returns a dataset that is the union of the results of the original first query and the results of the injected second query. One example usage of this multiple attacks is where the attacker uses the logically incorrect query attack to data about a table's structure then use the union query to get data from this table.

Example: an attacker could inject the text "' UNION SELECT card No from Credit Cards where acct No=10032 - -" into the login field, which produces the following query:

SELECT accounts FROM users WHERE login='' UNION

SELECT card No from Credit Cards where

acct No=10032 -- AND pass='' AND pin=

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "Credit Cards" table. In this case, the database would return column "card No" for account "10032." The database takes the results of these two queries, unions them, and returns them to the application. In many applications, the effect of this operation is that the value for "card No" is displayed along with the account information. [27]

IV. Piggy Backed Queries

Piggy-backed queries are the type of SQL injection queries formed when the hackers insert additional queries to be executed by the database, along with the intended query. The result will be cause extracting data; Adding or modifying data; Performing DOS; executing remote commands. This type of SQLIA is code injection category by injecting additional SQL queries with manipulating operation like “INSERT”, “UPDATE”, and “DELETE” clause that intend to modify database immediately following a legal SQL query. Vulnerability of this type of attack is that underlying back-end database must allow multiple SQL queries execute in a single string .it can be seriously harmful because it allows the commands to execute any SQL queries, even stored procedures attacks.

Example:

Query = Select * FROM employee;

Malicious insert another query:

Modified Query= SELECT * FROM employee; Drop table employee;

The result will be all valuable data of employee table that will be erased from database.

After completing the first query, the database would recognize the query delimiter (“;”) and execute the injected second query. The result of executing the second query would be to drop table users, which would likely destroy valuable information. Other types of queries could insert new users into the database or execute stored procedures. Note that many databases do not require a special character to separate distinct queries, so simply scanning for a query separator is not an effective way to prevent this type of attack. [11, 12]

V. Inference

Attack Intent : Identifying inject able parameters; extracting data; determining database schema. In this attack, the query is modified to recast it in the form of an action that is executed based on the answer to a true/-false question about data values in the database. In this type of injection, attackers are generally trying to attack a site that has been secured enough so that when an injection has succeeded, there is no usable feedback via database error messages. In this situation, the attacker injects commands into the application and then observes how the application responds. From careful observation, the attacker can deduce not only whether certain

parameters are vulnerable, but also additional information about the values in the database. There are two well-known attack techniques that are based on inference:

Blind Injection: Information is inferred from the behavior of the page by asking the server true/-false questions. If the injected statement evaluates to true, the site continues to function normally. If the statement evaluates to false, although there is no descriptive error message, the page differs significantly from the normally-functioning page.

Timing Attacks: A timing attack allows an attacker to gain information from a database by observing timing delays in the response of the database. Attackers structure their injected query in the form of an if/then statement, whose branch predicate corresponds to an unknown about the contents of the database. Along one of the branches, the attacker uses a SQL construct that pause the execution for a known amount of time (e.g. the WAITFOR keyword). By measuring the response time of the database, the attacker can infer which branch was taken in his injection and therefore the answer to the injected question.

VI. Alternate Encodings

Attack Intent: *Evading detection* .In this attack, the injected text is modified so as to avoid detection by defensive coding practices and also many automated prevention techniques. This attack type is used in conjunction with other attacks. In other words, alternate encodings do not provide any unique way to attack an application; they are simply an enabling technique that allows attackers to evade detection and prevention techniques and exploit vulnerabilities that might not otherwise be exploitable. These evasion techniques are often necessary because a common defensive coding practice is to scan for certain known "bad characters," such as single quotes and comment operators.

To evade this defense, attackers have employed alternate methods of encoding their attack strings (e.g., using hexadecimal, ASCII, and Unicode character encoding). Common scanning and detection techniques do not try to evaluate all specially encoded strings, thus allowing these attacks to go undetected. An effective code-based defense against alternate encodings is difficult to implement in practice because it requires developers to consider of all of the possible encodings that could affect a given query string as it passes through the different application layers. Therefore, attackers have been very successful in using alternate encodings to conceal

their attack strings. Note that these attacks may have no SQL keywords embedded as plain text, though it could run arbitrary SQL

Example: Every type of attack could be represented using an alternate encoding; here we simply provide an example of how mystic an alternatively-encoded attack could appear.

Username: "legalUser'; exec(0x73687574646f776e) - - "

Query: *SELECT name from authors where username = 'legalUser'; exec(0x73687574646f776e) - - AND password=' ';*

The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the string "SHUTDOWN." Therefore, when the query is interpreted by the database, it would result in the execution, by the database, of the SHUTDOWN command.

Query: *SELECT name from authors where username = 'legalUser'; exec(SHUTDOWN) - - AND password=' ';*

VII. Stored Procedures

Attack intent: performing privilege escalation; performing DOS; Executing remote commands. These attacks attempt to execute database stored procedures. SQL Injection Attacks of this type try to execute stored procedures present in the database. Most vendors ship databases with a standard set of stored procedures that extend the functionality of the database and allow for interaction with the operating system. Therefore, once the attacker determines the database type (potentially using illegal/logically incorrect queries) and then uses that knowledge to determine what stored procedures might exist. SQL Injection Attacks can be crafted to execute stored procedures provided by that specific database. Additionally, because stored procedures are often written in special scripting languages, they can be susceptible to privilege escalation, buffer overflows, and even provide access to the operating system.

Here is a stored procedure that checks credentials:

CREATE PROCEDURE DBO.is Authenticated
@userName varchar2, @pass varchar2, @pin int
AS EXEC ("SELECT accounts FROM users
WHERE login='" +@userName+ "' and pass='" +@password+ "' and pin=" +@pin);GO

Example:Notable example of SQL injection in Microsoft SQL Server

Microsoft SQL Server 2000 prior to service pack 3 contained stored procedure sp_MSdropretry which was publicly executable. This procedure's code looks like this:

```
CREATE PROCEDURE sp_MSdropretry (@tnamesysname, @pnamesysname)as declare
@retcodeint
/*
** To public
*/
exec ('drop table ' + @tname)
if @@ERROR 0 return(1)
exec ('drop procedure ' + @pname)
if @@ERROR 0 return(1)
return (0)
```

Note that the @tname parameter passed to the EXEC is not sanitated. This allows to exploit the injection by the db_owner user in any database:

```
execsp_executeSQLN'create view dbo.test as select * from master.dbo.sysxlogins'
exec sp_msdropretry 'anything update dbo.test set xstatus=18 where name= SUSER_SNAME()',
'anything'
exec sp_executeSQLN'drop view dbo.test'
```

VIII. Combination Attacks

Many attack vectors may be employed in combination:

- ✓ learn information useful in generating additional successful injections (illegal/logically incorrect)
- ✓ gain access to systems other than the initial database accessed by the application (stored procedures)
- ✓ evade detection by masking intent of injection (alternate encoding)

2.11. Related work

The first public discussions of SQL injection started appearing around 1998[23]JeffForristal, also known by the alias Rainforest Puppy, was one of the first people to ever document SQL injection. Forristal wrote the first public discussion about it, back in 1998.Over 15 years after it

was first publicly disclosed, SQL injection is still the number one threat to websites. Different researchers have used number of techniques in the prevention of SQL injection attacks. This chapter reviews different researcher work based on SQLIA Prevention and detection Techniques.

2.11.1. Prevention andDetection Techniques

Encryption and Tokenization Technique[33] the proposed system study on how to detect and prevent SQL injection attacks on web applications using encryption and tokenization technique. The tokenization process is applied on the input query by detecting spaces, single quotes and double dashes etc. This process converts the input query into fruitful tokens and that are stored in a dynamic table at the client side. The table name, field name and data are encrypted using AES algorithm .Encrypted the original input query and the tokenized table are sending to the server side. At the server side, input query is decrypted and in turn converts into various token which are stored in to another dynamic table. Both dynamic tables are compared and if both are equal, it seems that there is no injection attacked in the given query ,hence the query is proceed further to main database for retrieving result .If they are different, query is rejected and not forwarded to the database server. The Customized error message notification is given to the client. This paper has presented a lightweight method to prevent SQL injection attacks by applying query tokenization technique to convert SQL queries into number of useful tokens and then encrypting the table name, fields, literals and data on the query using AES Encryption algorithm.

SQLrand is an approach based on instruction-set randomization. [36] SQLrand provides a framework that allows developers to create queries using randomized instructions instead of normal SQL keywords. A proxy filter intercepts queries to the database and de-randomizes the keywords. SQL code injected by an attacker would not have been constructed using the randomized instruction set. Therefore, injected commands would result in a syntactically incorrect query. While this technique can be very effective, it has several practical drawbacks .First, since it uses a secret key to modify instructions, security of the approach is dependent on attackers not being able to discover the key. Second, the approach imposes a significant infrastructure overhead because it requires the integration of a proxy for the database in the system.

Convert plain text inputs into prepared statements.[37] This report presents a "code reengineering" that implicitly protects the applications which are written in PHP from SQL injection attacks. It uses an original approach that combines static as well as dynamic analysis. In this report, the researcher mentioned an automated technique for moving out SQL injection vulnerabilities from Java code by converting plain text inputs received from users into prepared statements.

Sayyed Mohammad SadeghSajjadi and BahareTajalliPour**"Preventing SQL Injection Attacks in Stored Procedures",**[38]they provided a novel approach to shield the stored procedures from attack and detect SQL injection from website. This method combines runtime check with static application code analysis so that they can eliminate vulnerability to attack. The key behind this attack is that it alters the structure of the original SQL statement and identifies the SQL injection attack. The method is divided in two phases, one is offline and another one is runtime. In the offline phase, stored procedures use a parser to pre-process and detect SQL statements in the execution call for runtime analysis. In the runtime phase, the technique controlled all runtime generated SQL queries related with the user input and checks these with the original structure of the SQL statement after getting input from the user. Once this technique detects the malicious SQL statements it prevents the access of these statements to the database and provides details about attack

DebabrataKar and SuvasiniPanigrahi[40]proposed approach to prevent SQL injection attacks by a novel query transformation scheme and hashing. They used a novel query transformation scheme that transforms a query into its structural form instead of the parameterized form. In order to store the transformed queries, they proposed to apply a suitable hashing function to generate unique hash keys for each transformed query. This approach can also be easily implemented on any language or database platform with little modification. However, this approach cannot prevent second order SQL injection attempts since the parameter values (especially the string values) are removed during the transformation process. Another drawback is that for query transformation, they used query transformation scheme lookup table.

SQL Injection Attacks: Techniques and Protection Mechanisms:[41]Code injection attack, particularly SQLIA is one of the scandalous issues. Controlling the pernicious SQL code/script on the web application and keeping up the end security is still a key test for the web engineer.

Web designers included in creating sites should consider these issues utilizing databases. This paper depicts how an assailant can abuse the web application by utilizing SQL injection attack to get private data from a database. Diverse assurance systems against SQLIA are likewise proposed.

A Survey of SQL Injection Countermeasures:[29,40] The various SQLIA types were studied and the prevention techniques which the researcher proposed went under a survey in this paper. This document discusses in detail the common 'SQL injection' technique, as it applies to the popular Microsoft Internet Information Server/Active Server Pages/SQL Server platform. It discusses the various ways in which SQL can be 'injected' into the application and addresses some of the data validation and the isolation issues of database identified with these attacks.

SQL Injection Attacks in Web Application:[30]this paper exhibits the different diverse procedures of SQLIA. By utilizing these methods the software engineers and framework managers can comprehend the SQLIA all the more altogether and secure the web application from SQLIA. However as the innovation keeps on growing, so will the security dangers and systems utilized by pernicious clients. As the clients of the web move their delicate information into the online environment, it is essential that security be given the most striking in the improvement of web applications.

WEBSSARI detects input-validation related errors using information flow analysis [42]. In this approach, static analysis is used to check taint flows against preconditions for sensitive functions. The analysis detects the points in which preconditions have not been met and can suggest filters and sanitization functions that can be automatically added to the application to satisfy these preconditions. The WEBSSARI system works by considering as sanitized input that has passed through a predefined set of filter. In their evaluation, the authors were able to detect security vulnerabilities in a range of existing applications. The primary drawbacks of this technique are that it assumes that adequate preconditions for sensitive functions can be accurately expressed using their typing system and that having input passing through certain types of filters is sufficient to consider it not tainted. For many types of functions and applications, this assumption is too strong.

Livshits and Lam use static analysis techniques to detect vulnerabilities in fifty software's. The basic approach is to use information flow techniques to detect when tainted input has been used

to construct an SQL query. These queries are then fagged as SQLIA vulnerabilities. The authors demonstrate the viability of their technique by using this approach to find security vulnerabilities in a benchmark suite. The primary limitation of this approach is that it can detect only known patterns of SQLIAs and, because it uses a conservative analysis and has limited support for un tainting operations, can generate a relatively high amount of false positives.

2.11.2. Preventive Coding Techniques

Protecting the web applications from SQL Injection, like most vulnerability, relies on

Defense-In-Depth: analyzing the state of security by performing a vulnerability assessment or pen test and regularly performing them after changes are made to the infrastructure. Web applications need to be developed using best practice techniques.

Sanitize Input: As far as possible use and Dark Side of SQL Injection allow only alphanumeric input. If there is a need to include punctuation marks of any kind, convert them into HTML Encoding.

Prepared Statements: Send precompiled SQL statements with one or more parameters. Parameter placeholders in a prepared statement are represented by the "?" and are called bind variables. Prepared statements are typically immune to SQL Injection attacks as the backend database will use the value of the bind variable exclusively and not interpret the contents of the variable in any other way.

Use Least Privilege: Give an application user the minimum rights on the database server it needs. Restrict the application user from permissions to access system stored procedures.

Custom Errors: Return SQL Query errors with custom nondescript responses. By not revealing any information in error codes, it is much more difficult for attackers to map the schema. Default error reporting for some frameworks includes developer debugging information.

2.12. Summary

There are systems that have been developed to detect SQL-I vulnerabilities such as Sania.[44] Sania identifies vulnerabilities during development and debugging phases by examining the SQL queries that occur between a web application and the database, and generating elaborate attacks according the vulnerabilities it finds.There has also been work on tools such as AMNESIA which is a runtime SQL-I detection tool and JCrasher which generates test cases, both of which are based in the Java language. [45]Aside from these two tools, there are many other tools both academic and commercial, such as Code Scan Labs: SQL-Injection, that identify SQL-I attack vulnerabilities.[46] However, this paper will focus on protecting SQL-I attack vulnerabilities during development of website by including filtering algorithm .

CHAPTER THREE: RESEARCH METHODOLOGY

3. Research design

This chapter describes and motivates the choice of research methodology, data generation method and how the data was analyzed. The outcome from the review is presented in this thesis. The data generation method questionnaire was chosen and used in this thesis to gather organizations vulnerability for SQLI attack which, awareness of SQLI, and find an alternative solution to SQLI attack. Finally we provided the proposed architecture with implemented techniques.

3.1.Research Methodology

3.1.1. Literary review

A proper literary review using *Google scholars, research gate and others* was conducted for two reasons; (1) to read the previous research studies in the area of IT-security and the subject of SQLI to examine what has already been written, and (2) to increase the author's knowledge in the subject area. Google Scholar provides a simple way to broadly search for scholarly literature. From one place, we can search across many disciplines and sources: articles, theses, books, abstracts and court opinions, from academic publishers, professional societies, online repositories, universities and other web sites. Google Scholar helps you find relevant work across the world of scholarly research.

3.1.2. Questionnaire

The data generation method questionnaire was chosen and used in this thesis. The reason for this was because a large number of people can be reached relatively easily and economically.To ensure the quality of the responses, the questionnaire was sent out to appropriate personnel at each company and organization that was contacted, for example some respondents were ICT-directors, heads of IT-managers and programmer. In the cases where companies and organizations didn't have any IT-department (usually because of a low number of employees), the questionnaire was sent to the chief executive officer (CEO) or the owner. The questionnaire was constructed in 2 different sections. Each section is inquiring on different types of subjects.

The first section is divided in two different parts and has questions about the website and quality standards or certifications in IT-security. The aim of this section was to receive general information about their website and security certifications. The second section is the main part of the questionnaire and it is composed of questions about SQLI such as if the companies and organizations are aware of it, what they are doing to prevent it and if anyone has ever made any attack attempts using SQLI The main goal in this section is to see if they are aware of SQLI and if they have been attacked. If they had been attacked a question about what changes that they made afterwards was asked.

3.2. Sampling Techniques

To study the current situation in the organization we used a purposive sampling technique because Purposive sampling (also known as judgment, selective or subjective sampling) is a sampling technique in which researcher relies on his or her own judgment when choosing members of population to participate in the study. Purposive sampling is a non-probabilitysampling method and it occurs when "elements selected for the sample are chosen by the judgment of the researcher. Researchers often believe that they can obtain a representative sample by using a sound judgment[9].ICT experts were purposively chosen from selected organization in order to investigate existing SQLI vulnerability in the organization.

Purpose of selection	**No. of selected**
Private organization	8
Government organization	10

Table3.2: Sample selection

3.3. Analysis Technique

This research aimed to answer four research questions the first three questions enable to understand the problem domain and the last one for the solution stack. To answer these questions using data gathering tools like qusetionary have used to collect data from different organization. These data were organized, analyzed and summarized for a better understanding of the SQLI

vulnerability cause, effect and preventing methods.The data generated from the questionnaire was summed up and represented in percentage

3.4.Result and discussion

The results obtained from questionnaire were presented in this section. The questionnaire was sent to 18 companies and organizations, with a response rate of 83 percent, 15 companies and organization answered. Some of the questions in the questionnaire were multiple-choice questions, which the percentage was calculated by dividing the answers with the total companies and organizations that answered the question.

NO	Question	Yes	No	Yes	No
1	Does your company/organization have a website?	15	0	100%	0%
2	Did your company/organization develop your website?	3	12	20%	80%
3	Do you hold information security certification(s) against any recognized quality standards by a accredited third party body?	2	13	13.3%	86.7%
4	Is your company/organization aware of SQL-injections?	6	9	40%	60%
5	Has anyone tried to use SQL-injection attack against your website?	4	2	66..6%	33.3%
6	Did the intruder(s) manage to access, modify or delete some confidential data?	2	4	66.6%	33.3%
7	Did you make changes to your website or database after the attack?	1	5	16.66	83.33%

Table 3.2 RESULTS OF SURVEY QUESTIONSECTION 1

Who developed the website?

20 percent had developed their website by themselves, and 80 percent state that someone else developed their website (Table 3.3).The 83.3 percent that in the previous question stated that they didn't develop their website by themselves answered the follow-up question about who develop their website, which is represented in Table 4.5. The great majority (46.7 percent) state that an outsourced company/organization developed their website, and the rest (20 percent) answered that an in sourced company/organization was responsible for the development.

Question	Outsourced company/organization	In sourced company/organization	Amateur web developer	Other	I don't know
Who developed your website?	7	1	2	2	3
Present	46.7%	6.7%	13.3%	13.3%	20%

Table 3.3 RESULTS OF SURVEY QUESTION 1.1.3

Awareness of SQL-injections

Only 40% companies and organizations were aware of SQL-injections, 60% percent weren't aware of SQL-injections (Table 3.4).

Question	Yes	No
Is your company/organization aware of SQL-injections?	6	9
Present	40%	60%

Table 3.4RESULTS OF SURVEY QUESTION 3.1

SQL injection attck attempt

all the companies and organizations that answered the question if anyone had tried to use SQL-injection attack against their website (Table 3.5), 26.66percent have noticed attack attempts against their website, 13.33% percent have not noticed any attempts, and 60%

percent are stating thatthey didn't know if someone had tried SQL-injections against their website.

Question	Yes	No	I don't know
Has anyone tried to use SQL-injection attack against your website?	4	2	9
Present	26.66%	13.33%	60%

Table 3.5RESULTS OF SURVEY QUESTION 3.4

Detect the attack

Table 3.5 represents how the attack was detected by the companies and organizations that had been exposed to SQL-injections. The most common way to detect the attack was afterwards in logging systems (33 percent), followed by intrusion detection system (16percent), and because the website was changed (16.66 percent). 16.66 percent state that the company and organization don't detect the attack.

Question	Intrusion detection system (IDS)	afterwards in logging systems	Because the website was changed	Deleted database	We did not detect the attack	Other	I don't know
How did you detect the attack?	2	2	1	0	0	0	1
Present	33.33%	33.33	16.66	0	0	0%	16.66%

Table 3.6 RESULTS OF SURVEY QUESTION 3.5

Preventing SQL- injections

When asked the multiple-choice question how the companies and organizations are preventing SQL-injections, 6.7 percent state that they validate the input on the server, 20 percent state that they validate the input on the website, and 73.3 percent state that they are doing nothing (Table 3.6).

Question	Validation of input on the website	Validation of input on the server	I don't know
How is your company/organization preventing SQL- injections?	3	1	11
Present	20%	6.7%	73.3%

Table 3.7 RESULTS OF SURVEY QUESTION 3.2

Protect confidential data

The companies and organizations were asked how they protect the confidential data that they store in databases (Table 3.7). 53.4 percent of them are using some kind of encryption, 13.3 percent is using salted hash, and 13.3 percent is doing nothing. 20 percent state that they don't know.

Question	Encryption	Salted hash	We are doing nothing	I don't know
How is your company/organization protecting confidential information such as passwords and credit card details?	8	2	2	3
Present	53.4%	13.3%	13.3%	20%

Table 3.8 RESULTS OF SURVEY QUESTION 3.3

3.5. Conclusion

The questionnaire was sent to employees in companies and organizations that were working in

the IT-department.In the cases were it didn’t exist an IT-department, most often the owner responded to the questionnaire. If the questionnaire was sent to someone who wasn’t aware of SQL-injections, they would skip all questions about that topic. This assures that the answers about SQL-injections were answered by someone who had knowledge aboutthem.Companies and organization were aware of SQL-injections answered the all question . the companies and organizations that answered the question if anyone had tried to use SQL-injection attack against their website 26.66 percent have noticed attack attempts against their website, 13.33% percent have not noticed any attempts, and 60% percent are stating thatthey didn’t know if someone had tried SQL-injections against their website. we conclude that most companies and organizations not doing enough to prevent SQL-injections.

CHAPTER 4: DESIGN OF THE PROPOSED SQLI-A PREVENTION SYSTEM

4. Introduction

In this Chapter we have presented the design of the Proposed SQLIA prevention System. Different components of the Proposed SQLIA prevention System are described with their relevance and techniques to use while building those components. The Chapter presents the architecture with implemented techniques.

4.1.Requirements of prevention System

To build an SQIL-A prevention System, a certain requirement must be fulfilled to make the system efficient. Before describing those requirements, it is necessary to introduce four important terms that clearly define the requirements [20]:

- ✓ ***False Positive (FP***): represents the pattern which are classified as SQLI by the prevention system but they are normal.
- ✓ ***True Positive (TP):*** represents pattern which are classified as SQLI by the prevention system and they are truly SQLI.
- ✓ ***False Negative (FN***): represents the pattern that are classified by the prevention system as being legitimate when in fact they are SQLI
- ✓ ***True Negative (TN):*** represents the number of instances that are classified by the prevention system as being legitimate and that really are truly legitimate.

In order to make the SQLI-A helpful it should have to fulfill the following requirements [44].

Accuracy: this property ensures that the prevention system does not classify legitimate instances as anomalous.

Performance: it must be able to classify the traffic without adding a noticeable overload to the network. The performance of the prevention system is measured using confusion matrix including training and testing time.

Completeness: this property is the core of the prevention system. It states that an prevention system should be able to detect all SQLI-A. In practice, this property is very hard to achieve because the prevention system must be able to detect known attacks as well as the new ones.

4.2. System Overview

The main goal of this research is to design SQLIA prevention System using filter approach .This work is proposed in order to prevent SQLIA. One of our main focuses for this project was to create a system architecture that required minimal changed to an already deployed web application that was connected with a database. To simulate this environment we developed a vulnerable web application that was connected with a database. The system consists of a Apache web server, a web application that was built on PHP and a MYSQL database. The web application has two different pages. A login page that requires username and password to login and a welcome screen that shows up after a successful login. The database has a user table which consists of username and password. With these components .We simulated an already deployed system that was functioning for the purpose it was built. To minimize the change that has to be done on the system we had to place the filter program in a spot that wouldn't require many changes to the current code. The web server should forward the entire http request to the filter program. The http request would then be sent back to the main page (web server), which would then send the SQL query to the database. The steps in which the SQL-Injection prevention system summarized as follows:

- ✓ Construct a list of common SQL-Injection commands or pattern identification.
- ✓ Prevent a SQL-Injection attack to a database using developed filter program.
- ✓ Prove that SQL-Injection can be prevented using the filter developed to work on the PHP.

SQLLA Pattern Identification

The attacker primary technique of SQLI is finding a signature pattern to attack. A pattern is an order of characters that will always appear in the request for that particular attack type or the signature of the attack. After extracting a signature for the known SQLI attacks, then uses the signatures to attack the websites.

4.3.System Architecture

In this Section, we have proposed architecture for SQL-I attack vulnerability prevention system that can filter the SQL-I attack. The architecture of the system is illustrated in Figure 4.1.In a client server model, a filter program that runs on the application server placed in between the client and the server. The presence of the filter program is not known to the user.

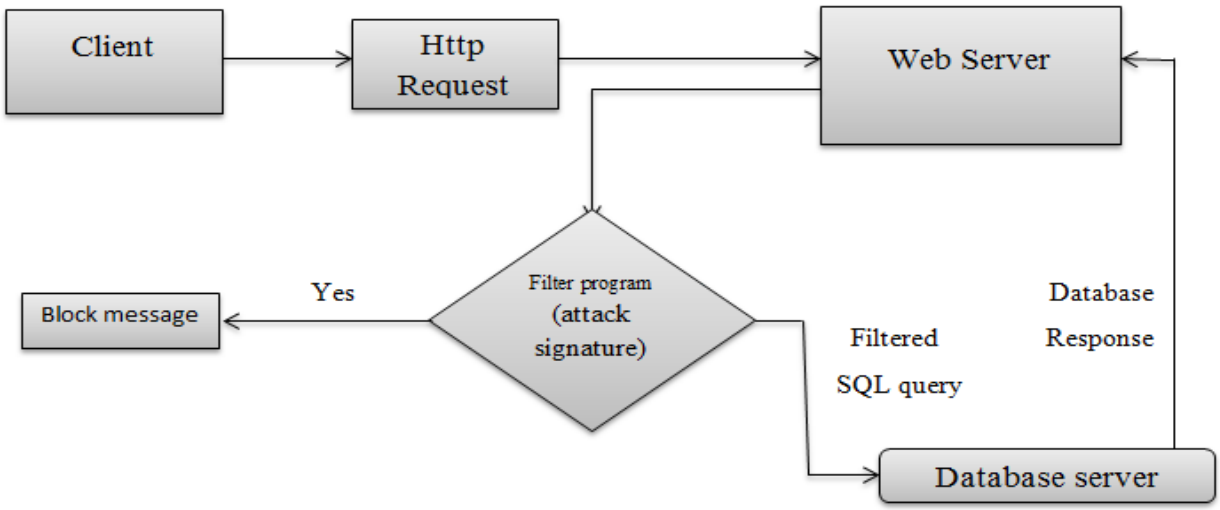

Figure 4.1: Proposed Architecture for SQLI-A prevention System

4.4.Component of proposed system

Client: desktop computer or workstation that is capable of obtaining information and applications from a server. Client sends HTTP request to web server. The request can be free from the attack patterns or may contain an attack pattern. However the clients send a request and wait for responses for that request. The response of the web server can be either positive response or negative response (block) as shown in Figure 4.2

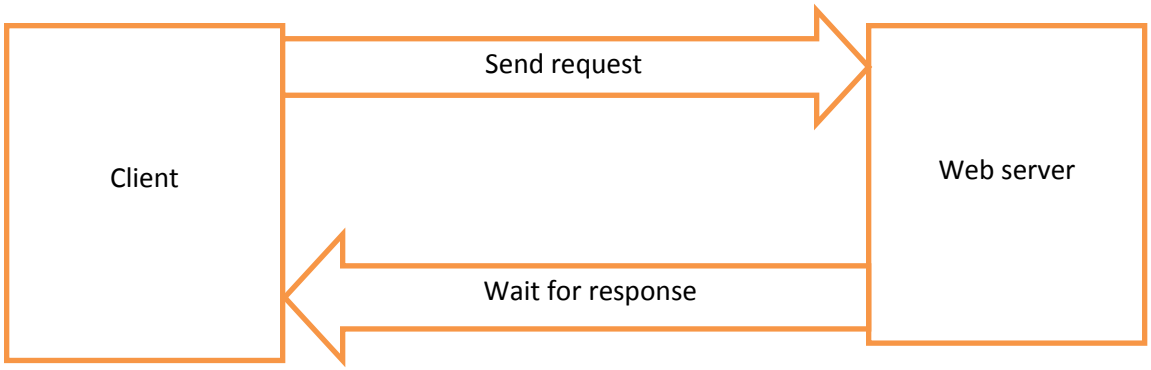

Figure 4.2: the general architecture of clients

Http request: The Hypertext Transfer Protocol (**HTTP**) is designed to enable communications between clients and servers. **HTTP** works as a **request-**response protocol between a client and server. A web browser may be the client, and an application on a computer that hosts a web site may be the server.

In general, HTTP carries data from the clients to the web server using predefined rules (protocols) which enables the exact communication between them.

Web server: Web Server is designed to serve HTTP Content. Web Server is mostly designed to serve static content, though most Web Servers have plug-in to support scripting languages like Perl, PHP, ASP, JSP etc. through which these servers can generate dynamic HTTP content.

Thus, web server accepts the requests send from clients and filter the requests and respond accordingly. Which means if the request contains an attack patterns, it will block the request; else it will response based on the requested contents. In general, the web server implements two algorithms; filter algorithm and block algorithm.

Filter algorithm: proposed filter program runs on the server before the requested (URL pattern) page with which it is associated filters SQL injection patterns, if user submits such pattern. Itblocks the request and display error message to the user. On other hand if the user input doesn't include SQL injection pattern, it redirect the request to the database and responds.

In general the algorithm has four sub-components as shown in Figure 4.3.

Accept request from the client: this component is ready to get any HTTP request from any clients. So, with the help of this component, the requests send from the clients will be accepted.

Get attack patterns: this component will accept all attack patterns and extract all SQLI attacks signatures from. The attack patterns are listed below.

1. Tautologies Signature
   - *(') followed by OR operator*
2. Illegal/logically incorrect queries Signature

- *('), AND operator, ORDER BY, HAVING clause*

3. Union Query UNION Signature:

- UNION

4. Piggy-Backed Query Signature:

  - (;) SQL command terminator

5. Stored Procedure Signature:

  - (;)SQL command terminator, EXEC

6. Blind SQLI Signature

  - AND operator and Conditional operator

7. Timing Attack Signature:

  - WAITFOR,IF,ELSE

8. Alternate encoding Signature:

Char(),ASCII(),HEX(),UNHEX()

Compare the request and the patterns: at this phase the two values (value from the client and value from the attack patterns) will be compared. This phase generates an identified data. Identified data in a sense, it may be attack pattern free or contains attack patterns.

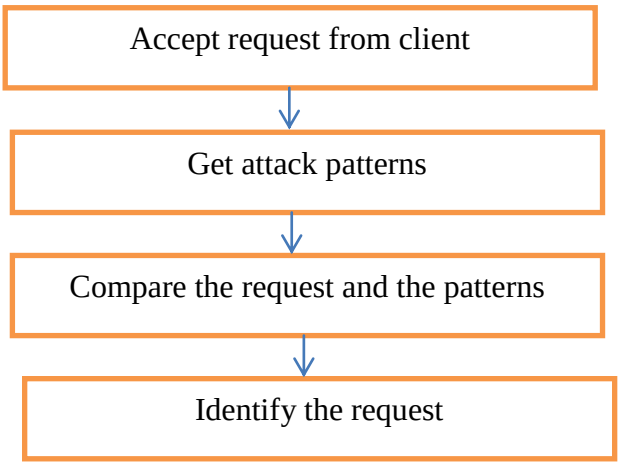

Figure 4.3: The general architecture of filter algorithm

Filter algorithm:

Get requests from client

While (request)

Get attack patterns

Compare the request and patterns

Identify the request

End while

Database server*:* the database containing content (e.g., news) and customer data (e.g., usernames and passwords, social security numbers and credit card details).

Block algorithm: this algorithm is responsible for blocking requests of the attackers (requests that contains attack patterns). In this step, the output of filter algorithm will be the input. So, it accepts the identified request and block based on the status of the request set by the filter algorithm as shown in Figure 4.4:

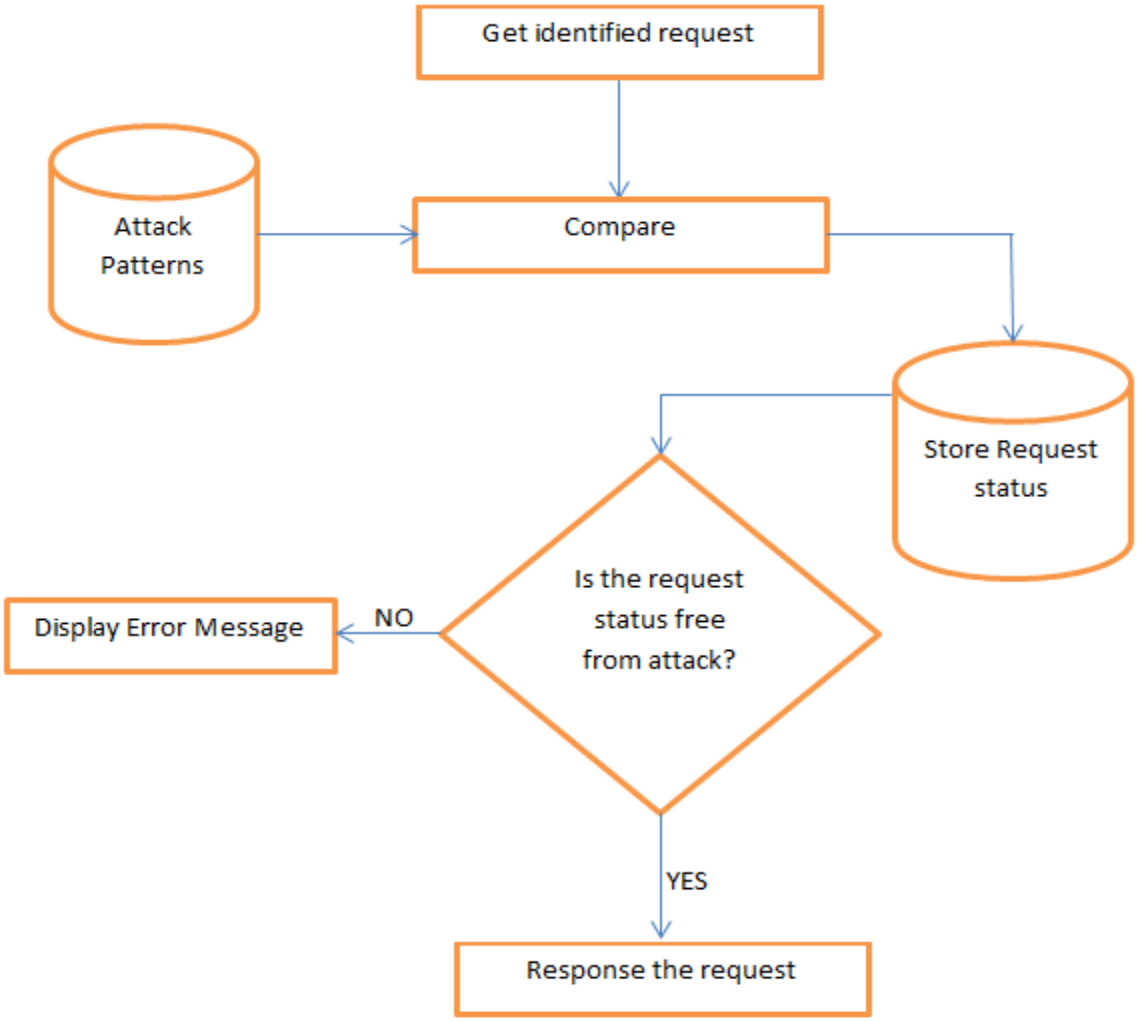

Figure 4.4: the general diagram of block sub-component

Block algorithm

Get identified requests

While (identified requests)

If (identified requests are free of attack patterns)

Response

Else

Block the request(display error message)

End if

End while

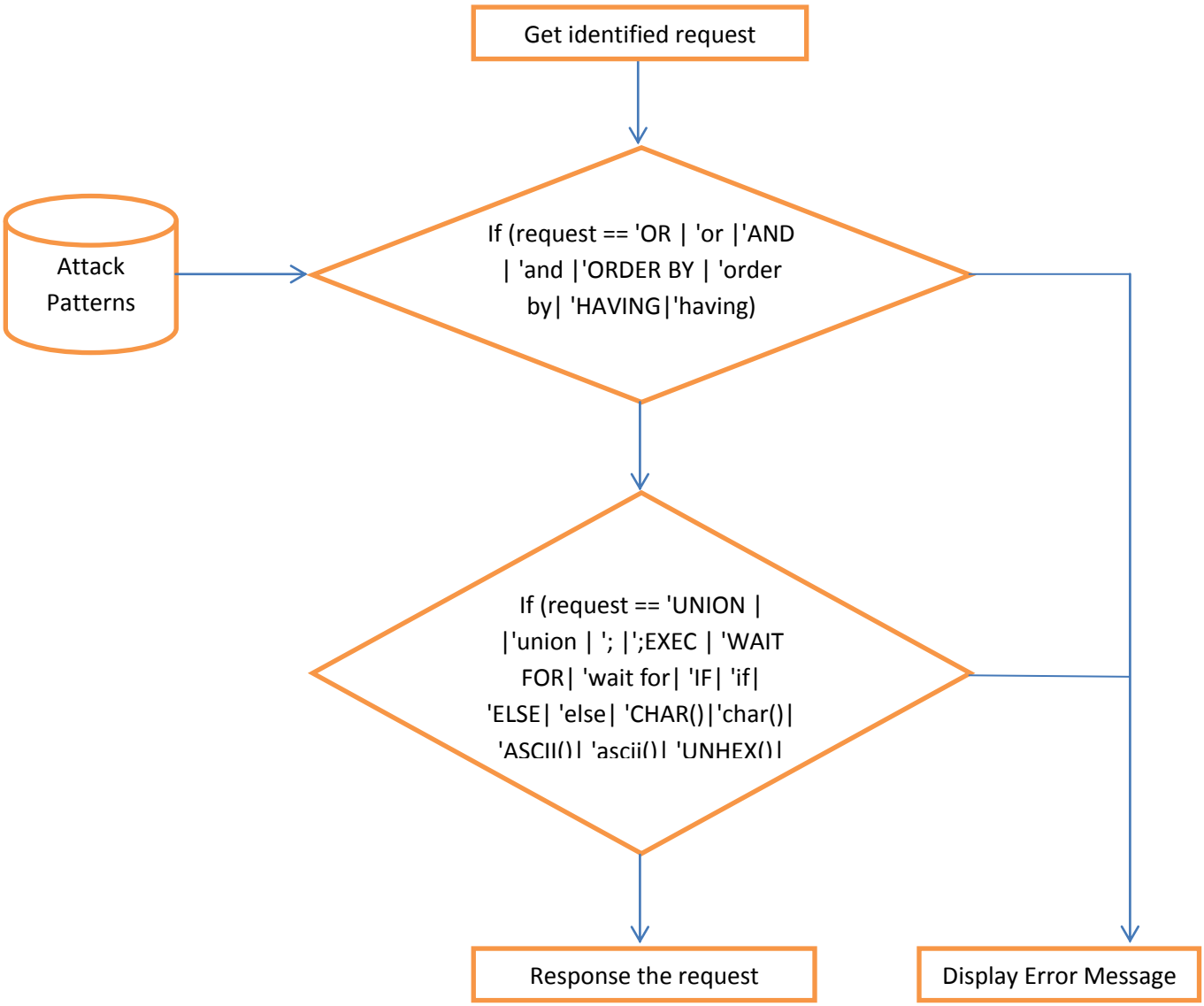

Figure 4.5 Detail of block algorithm sub-component

CHAPTER FIVE: IMPLEMENTATION AND EXPERIMENT

In this Chapter the implementation detail for SQL-I attack vulnerability prevention system will be presented. We have used this implementation to evaluate the proposed work which is the filtering technique for securing website from SQL-I attack and the evaluation will be presented here. In this Chapter we will cover overview of how the implementation is done, tools used to do this work, and finally measure the performance of the work done.

5.1. Overview

The main goal of this research is to study how prevent website from SQLIA using filtering techniques. This work is proposed in order to prevent known SQLIA patterns. The flow of this work starts by installing WAMP server;build filtering techniques whichprevent website from SQLIA vulnerable. Finally we test the performance of proposed system using different test cases.

5.2. Experiments and Results

The Experiments in this thesis was demonstrated using WAMP server 2.4 and PHP as server scripting language on Windows 7 platform. The case studies are done by injecting SQL code into the login form on the website. The purpose of the injected SQL code is to manipulate the query sent to the database in order to gain access to the welcome page.

Case one: *authorized user interacts with user interface of web based applications using username and password*. The purpose of the case one is to get redirected to the welcome page from the login page by *authenticated login credentials*.

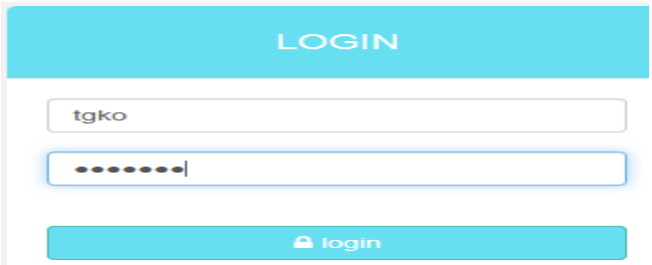

Figure 5.1 Login pages

The output is

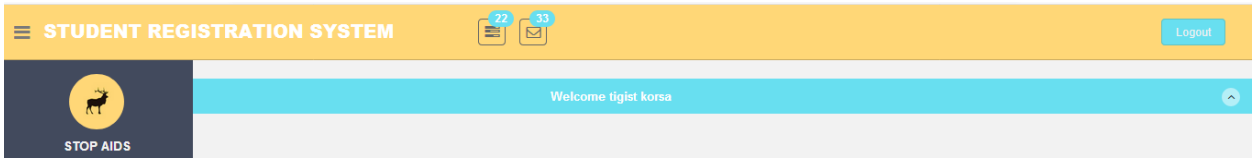

Figure 5.2 Redirect page after authorized user login

Case two: The purpose of the test is to get redirected to the welcome page from the login page by performing SQLI tautology attack. The SQL code used in this test should be the same as in case one. The web server not include filter program which directly connected with the database during this test. The test case are shown in table 5.1

SQL injection Tautology attack pattern			
No	**Username**	**Password**	**SQL Query**
1	' or '1'='1	' or '1'='1	SELECT * FROM users WHERE name='' or '1'='1' and password='' or '1'='1'
2	' or ' 1=1	' or ' 1=1	SELECT * FROM users WHERE name='' or ' 1=1' and password='' or ' 1=1'
3	' or ''='	' or ''='	SELECT * FROM users WHERE name='' or ''='' and password='' or ''=''
4	' or 1-- --	Empty	
5	' or '10'='7+3		
6	' or '10'='(SELECT 14/2) + 3		
7	' or true-- --	Not required	
8	' – '	' – '	
9	' & '	' & '	
10	'^'	' ^ '	

Table 5.1 SQL injection Tautology attack pattern

The main objective of this type of attack is to inject code in one or more conditional statements so that they are every time evaluated to be true. The consequences of this type of attack, is dependent upon how the results of the query are used within the application. The attack is successful, because he or she get redirected to the welcome page from the login page as legal user.One example from case two an attacker submits ***' or '1'='1*** for the login input field and password input field, the resulting query will be formed as:

$SQL =”SELECT * FROM users WHERE name='' or '1'='1' and password='' or '1'='1'”;

It is shown in the Figure 5.3 below.

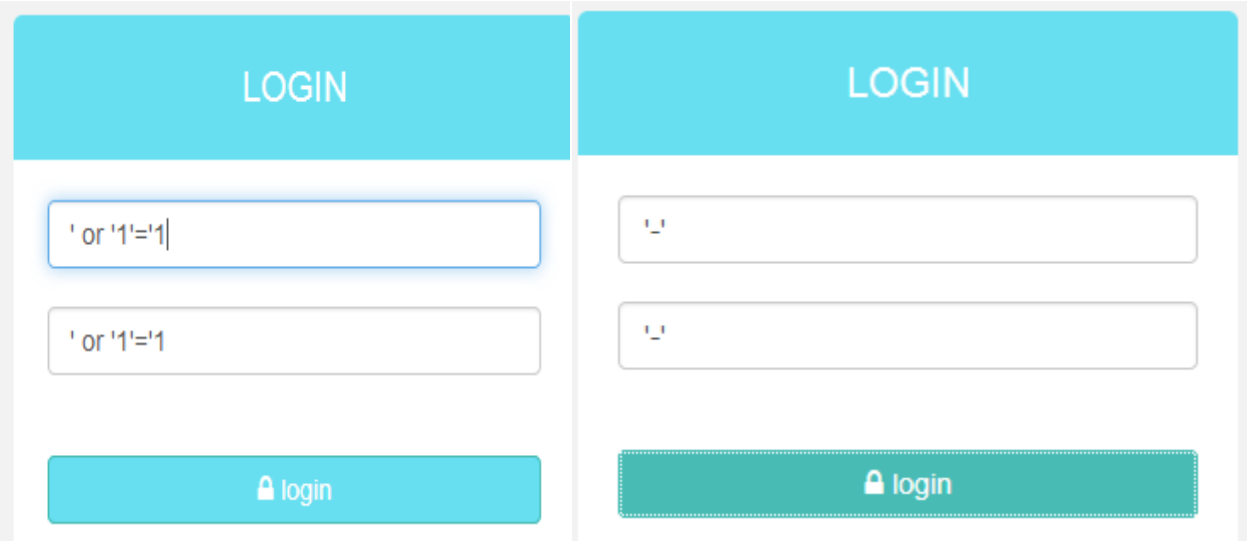

Figure 5.3: login page with SQLI input

The output is

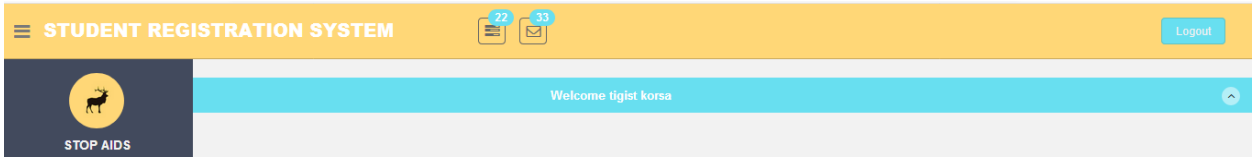

Figure 5.4: Redirected page after successfully login with injection input

Case three: The purpose of the test is to get redirected to the welcome page from the login page by performing SQLI tautology attack. The SQL code used in this test should be the same as in case two. The web server include filter program which directly connected with the database during this test. In this case the when user try to inject the code the filter program filter pattern and block user from getting access to the next page.

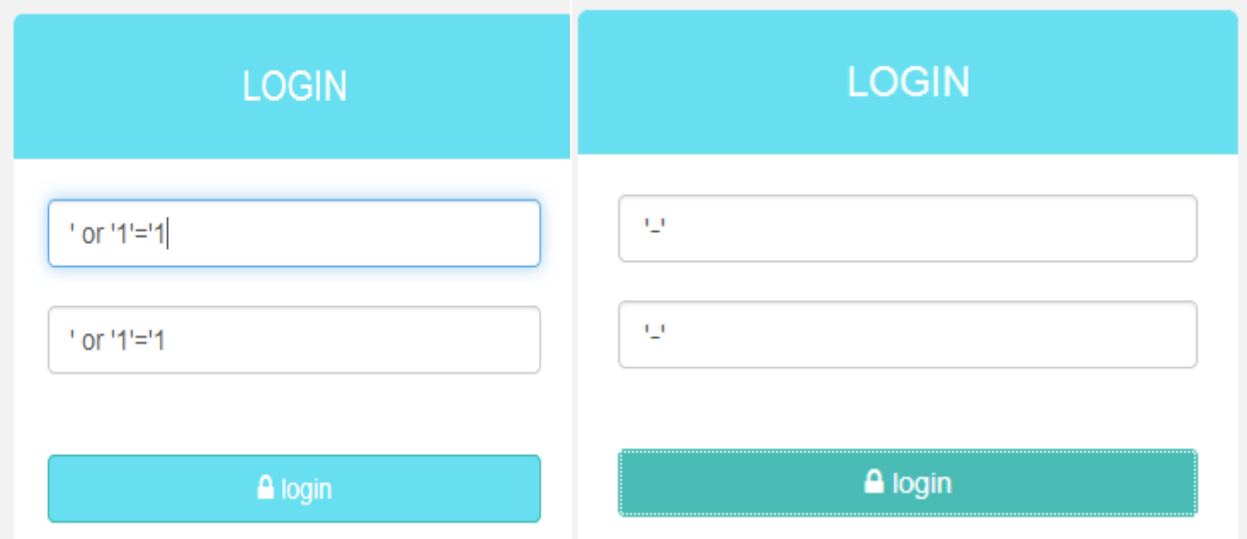

Figure 5.5: login page with SQLI input

The output is

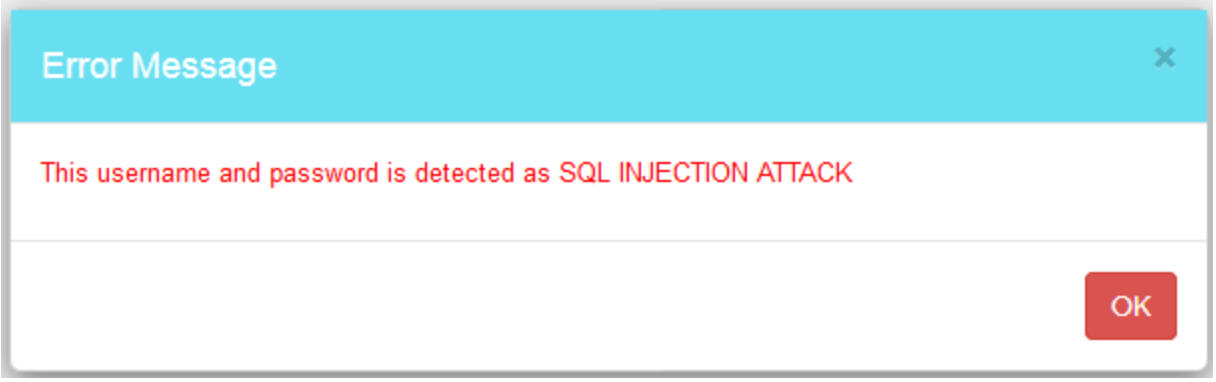

Figure 5.6: Error Page after unsuccessful SQL injection block by the filter

Case four: Illegal or logically incorrect queries

The immediate solution to such leakage of schema information is to suppress the error messages from getting echoed back to the client. Most web application languages and database platforms provide configuration settings to disable error messages. It is highly recommended that error messages are disabled on production servers.

SQL injection Illegal or logically incorrect queries attack pattern		
Username	**Password**	**SQL Query syntax error**
“abc' ABCD”	“abc”	Could not connect to data: You have an error in your SQL syntax; check the manual that corresponds to your **MySQL** server version for the right syntax to use near 'ABCDâ€• ' AND **password** = "'abc"" at line 1
abc' ORDER BY 4 -- '	“xyz”	Could not connect to data: Unknown column '4' in 'order clause

OR 'ABC' = CoNcAt('a', 'b', 'c') OR 'abc' <cOnCaT('D', 'E', 'F') OR ASCII('E') = ASCII('A') + 4	Any	You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'ABC' = CoNcAt('a', 'b', 'c')' AND password='OR 'ABC' = CoNcAt('a', 'b', 'c')" at line 1

Table 5.2 SQL injection Illegal or logically incorrect queries attack pattern

Case 5 Union Query

In SQL, the UNION keyword is used to combine the result from multiple SELECT statements into a single result set. In UNION-based SQL injection attacks, the attacker injects an additional query using the UNION keyword to bring data from other tables or columns into the result set so that they can be displayed on the web page.

For a UNION query to be valid, both queries must have the same number of columns of same data types. This information is previously collected by the attacker through error messages from illegal or incorrect query attacks. UNION queries can also be used to determine the number of columns in the SELECT list of the original query.

Case 6 Piggy-Backed Query

Piggy-Backed Query (Stacked queries) is possible only when the database server supports batched execution of queries, such as PostgreSQL and Microsoft SQL Server. If batched queries are supported by the backend database server, the attacker can virtually append any type of SQL command and get them executed along with the original query. Fortunately, MySQL database server does not support batched queries, so PHP/MYSQL applications are not vulnerable to this type of injection attacks

5.3.Performance Evaluation

In this section, we measure the performance of the proposed SQL injection prevention system and compare the result with excluding proposed SQL injection prevention system from web pages. To conduct this evaluation, we consider the requirements of Requirements of SQLIA prevention System. Therefore, the evaluation includes those measures like accuracy, performance, completeness and scalability. Standard metrics for measuring the performance of SQLIA prevention System is evaluated by computing measures from the value in the confusion matrix shown in Table 5.3.

		Actual class	
		Negative Class(normal)	Positive Class(Attack)
Predicted Class	Negative Class(normal)	True Negative(TN)	False Negative (FN)
	Positive Class(Attack)	False Positive(FP)	True Positive (TP)

Table 5.3: Confusion Matrix

5.3.1. Performance Evaluation: Accuracy

As mentioned earlier the effectiveness of SQLI-A prevention System is measured in terms of accuracy in which it identifies how much do the SQLI-A prevention System classify the coming user input as normal and attack. The accuracy of the proposed system is calculated using Equation 5.1.

- Accuracy:

$$\text{accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \quad (5.1)$$

For our case we get the accuracy SQLI-A prevention system. We used 10 attack cases as stated above and we got 10 attacks out of the given which means zero are classified as falsely as normal. And also from the normal 5 cases 4 is predicted normal and the rest 1 is predicted as attack falsely. From this we got TP =10, FP = 0, TN =5 and FN= 1 so we can calculate it accuracy and we got 93%.So from this we found that the accuracy of the proposed system is encouraging results.

5.3.2. Performance Evaluation: Performance

The main objective of this performance evaluation is to identify whether it add noticeableoverload or not to the SQLI-A prevention System. In particular, the following measures will be used to assess the SQLI-A prevention System'sperformance including accuracy:

5.3.3. Performance Evaluation: Completeness

The aim of this completeness is to minimize false alarm rate. When the false alarm rate decreases we can assume that the prevention system starts examining all SQLI-A pattern in user input. So for our case false alarm rate for the SQLI-A prevention system is less.so from this we can conclude that the proposed SQLI-A prevention system is better detecting SQLI-A pattern and it is complete.

5.4. Discussion

Based on the results we have obtained in the previous Chapter we present here our perception of the result in relation to the research objectives stated in Chapter 1. As a reminder, the primary objective of this work was to design SQL injection Vulnerability protection system to web security threats and that can prevent SQL injection attack and this thesis work answers the questions:

- ✓ What are currently and widely used SQL-I techniques?
- ✓ How can we minimize SQL-I vulnerabilities?
- ✓ How SQL-I attack Prevention is better than cure?
- ✓ Why SQL-I attack vulnerability prevention systems are needed in any networked organization?

After this research work and the experimentation ,we have obtained it accuracy was 93%.So from this we found that the accuracy of the proposed system encouraging results as we have assumed at the beginning of this work. In this thesis work, we have tried to develop filter algorithm that can detect SQL injection pattern and block those pattern before hacker get access to the database. We assume that it will increase the detection performance of our system and our result proves this expectation. We developed and tested difference types of SQL injection using numbers of test .We propose an optimization scheme that incorporates prediction confidence accuracy metrics. Compared to results using the proposed SQLI-A prevention system we obtained a promising result. The filter approach avoids memory requirements to store the legitimate query in repository and facilitates fast and efficient accessing mechanism with database. Our experimental results show that this approach can effectively prevent all types of SQL injection attempts. This approach does not require major changes to application code and has negligible effect on performance even at higher load conditions due to its low processing overhead.

CHAPTER SIX: CONCLUSION AND FUTURE WORK

6.1. Conclusion

SQLIA prevention system has improved with age, but this improvement seems to be a continuous process as advancement in the technology opens the door with a loop-hole for attackers every time.

In this thesis work, we have tried to develop a SQLI-A prevention system that tries to protect the vulnerability of websites from SQLI-A. In the present work, we discussed the design and development of "SQLIA prevention system" which is built on server side scripting in PHP. This component is test using vulnerable website and test cases. The thesis presented filter approach for detecting SQL injection pattern. The main contribution of this thesis work is to build an SQLIA prevention system that can examine SQL injection pattern attacks .We concludes that our filtering technique can prevents. SQL injection points to be vulnerable on the web application.

6.2. Further Work

While our work has produced some promising results, it is necessary to improve our system further to detect more known and unknown attacks. Our proposed system also needs further testing on a test case with more variety of attacks. Some potential future works that could be a continuation of our work is as follows:

- ✓ This work is done using filter approach which detect attacker pattern only block. But on the future prevention system it can provide action and gives alerts for the administrator.
- ✓ The other issue to increase the detection rate in filter approach is to add other perfect pattern matching algorithm to the filter first then add anomaly detection system to it.
- ✓ In this thesis work, even if we performed evaluation for the performance of its completeness and scalability; we could not observe that it is 100% complete and it needs more time. So it is also an open issue to develop more scalable and complete SQLI-A prevention system.

Reference

[1] KASPERSKY SECURITY BULLETIN 2015 https://securelist.com/kaspersky-security-bulletin-2015-overall-statistics-for-2015/73038/

[2] INFORMATION SECURITY BREACHES SURVEY 2015 | technical report Blue)https://www.pwc.co.uk/assets/pdf/2015-isbs-technical-report-blue-digital.pdf

[3] PhillipLoranger, DOEDCISO Robert Ingwalson, FSA (Federal Student Aid) CISO).

[4] Joseph Herbrandson," Most Common Attacks Affecting today's Websites."https://blog.sucuri.net/2014/11/most-common-attacks-affecting-todays-websites.

[5] Steven Cook,"A Web Developer's Guide to Cross-Site Scripting" GSEC Version 1.4b SANS Institute Info Sec Reading Room, January 11, 2003

[6] OWASP, the Open Web Application Security Project," The Ten most critical web application security risk," https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf

[7] William G.J. Halfond and Alessandro Orso "Detection and Prevention of SQL Injection Attacks" Georgia Institute of Technology {whalfond,orso}Sec.gatech.edu,2012

[8]Telstra Cyber Security Report "Managing risk in a digital world" technical report 2017

[9]John Wiley & SonsBlack,"Business Statistics: Contemporary Decision Making" 6th edition, , K. (2010)

[10] Christy, S. CWE/SANS Top 25 Most Dangerous Software Errors. Report No. 1.0.3. Common Weakness Enumeratinon (CWE) and System Administration, Networking, and Security Institute (SANS)(2011)

[11] Webpage:" Most Common Attacks affecting today's Websites" https://www.linkedin.com/pulse/most-common-attacks-affecting-todays-websites-james-fisher

[12] C. Anley. Advanced SQL Injection in SQL Server Applications. White paper, Next Generation Security Software Ltd., 2002

[13] C. Anley. (more) Advanced SQL Injection. White paper, Next Generation Security Software Ltd., 2002.

[14] webpage: "What is a web application attack and how to defend against it." https://www.acunetix.com/websitesecurity/web-application-attack/

[15] jayeetaMajumder&GargiSaha, "Analysis of SQL Injection Attack" Special Issue of International Journal of Computer Science & Informatics (IJCSI),ISSN (PRINT): 2231–5292, Vol.- II, Issue-1, 2

[16] "History of SQL Injection"[Online].Available:http://hackertarget.com/10-years-of-SQL-injection

[17] Justin Clarke, SQL Injection Attacks and Defense, Second Edition, Syngress Publication, July 2, 2012, ISBN-13: 978-1597494243

[18] Damele A G,"Advanced SQL Injection Whitepaper", Presented at BlackHat Europe 09. [Online].Available: http://www.blackhat.com/presentations/bh -europe-09/Guimaraes/Blackhat-europe-09-DameleSQLInjection-whitepaper.pdf

[19] N. W. Group. RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1.Request for comments, The Internet society, 1999.

[20] M. Dornseif. Common Failures in Internet Applications, May 2005.http:// md.hudora.de/presentation/2005-common-failures dornseif-common-failures-2005-05-25.pdf

[21] T. M. D. Network. Request.servervariablescollection.Technicalreport, Microsoft Corporation,2005.http://msdn.microsoft.com/library/default.asp?url=/library/ens/iissdk/html/9768ecfe-8280-4407-b9c0-844f75508752.asp

[22] S. McDonald. SQL Injection: Modes of attack, defense, and why it matters. White paper, GovernmentSecurity.org, April 2002.

[23] Clarke, J. (2012) SQL Injection Attacks and Defense. 2nd ed. Waltham, USA: Syngress.
Ezumalai, R. &Aghila, G. (2009) Combinatorial Approach for Preventing SQL Injection

Attacks. In: Proceedings of the 2009 IEEE International Advance Computing Conference, (pp.1212-1217).

[24] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso.A classification of sql injection attacks and countermeasures.14th ACM SIGSOFT international symposium on Foundations of software engineering, March 2006.

[25] M. Howard and D. LeBlanc. Writing Secure Code. Microsoft Press, Redmond, Washington, second edition, 2003.

[26] S. McDonald.SQL Injection Walkthrough.Whitepaper,Security Team, May 2002.http://www.securiteam.com/securityreviews/5DP0N1P76E.html.

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

[28]SruthyManmadhan and T. Manesh, "A Method of detecting SQL Injection attack to secure web applications," International journal of Distributed and Parallel Systems(IJDPS), vol. 3, no. 6, November2012.

[29] Dr.R.PMahapatra and mrssubikhan(2012), “A surveyofsql injection countermeasures”

[30] Mihir Gandhi, JwalantBaria (2013) “SQL INJECTION Attacks in Webapplication”,International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-6, January 2013

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

[33] SQL injection prevention technique using encryption, meharsood, SMITA SINGH International Journal of Advanced Computational Engineering and Networking ISSN: 2320-2106,Volume-5,Issue-7,Jul.-2017

[34]Boyd Stephen W. and KeromytisAngelos D. SQLrand: Preventing SQL injection attacks. The 2nd Applied Cryptography and Network Security (ACNS) Conference, pages 292–302, June 2004.

[35] Debar H., Dacier M., and Wespi A., "Towards a Taxonomy of Intrusion Detection Systems",Computer Networks, 31:805{822, April 1999}

[36]SQLrand:Preventing SQL Injection Attacks StephenW. Boyd and Angelos D.Keromytishttps://www.cs.columbia.edu/~angelos/Papers/SQLrand.pdf

[38] Sayyed Mohammad SadeghSajjadi and BahareTajalli Pour (2013) "Study of SQL Injection Attacks and Countermeasures",International Journal of Computer and Communication Engineering, Vol. 2, No. 5, September 2013

[40] DrR.P.Mahapatra and MrsSubi Khan (2012) "A Survey of SQL Injection countermeasures", International Journal of Computer Science & Engineering Survey (IJCSES) Vol.3, No.3, June 2012

[40] DebabrataKar, SuvasiniPanigrahi, Prevention of SQL Injection Attack Using Query Transformation and Hashing IEEE International Advance Computing Conference (IACC),2013

[41]Nikita Patel, Fahim Mohammed, SantoshSoni (2011) "SQL Injection Attacks: Techniques and Protection Mechanisms",International Journal on Computer Science and Engineering (IJCSE) ISSN: 0975-3397Vol. 3 No. 1 Jan 2011 199-203

[42] Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In Proceedings of the 12th International World Wide Web Conference (WWW 04), May 2004.

[43] V. B. Livshits and M. S. Lam. Finding Security Errors in Java Programs with Static Analysis. In Proceedings of the 14th Usenix Security Symposium, pages 271286, Aug. 2005.

[44]Kosuga, Y.; Kernel, K.; Hanaoka, M.; Hishiyama, M.;Takahama, Yu., "Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection, "Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-ThirdAnnual , vol., no., pp.107-117, 10-14 Dec. 2007

[45] Mei Junjin, "An Approach for SQL Injection Vulnerability Detection," Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on , vol., no., pp.1411-1414, 27-29 April 2009

[46] Webpage "About page for Code Scan from Code Scan Limited" http://www.codescan.com/about-codescan/what Retrieved on 2010-04-10.

[47]WebPages “SQL injection: attacks and defenses”, Dan Boneh https://crypto.stanford.edu/cs142/lectures/16-sql-inj.pdf

Appendix 1: Questionnaire

Section 1: Application Review and Standards

1.1. **Website – Questions about your website to learn more about when and by whom it was developed.**

1.1.1. Does your company/organization have a website?

- ☐ Yes
- ☐ No(skip to 1.2)

1.1.2. Did your company/organization develop your website?

- ☐ Yes (skip to 1.1.4)
- ☐ No

1.1.3. Who developed your website?

- ☐ Outsourcedcompany/organization
- ☐ Insourcedcompany/organization
- ☐ Amateur webdeveloper
- ☐ Other:_____
- ☐ I don’t know

1.1.4. When was the website developed?

- ☐ Less than 1 yearago
- ☐ 1-5 yearsago
- ☐ More than 5 years ago
- ☐ I don’tknow

1.2. Quality Standards and Certifications Questions about your companies/organization’s quality standards and certifications.

1.2.1. Do you hold information security certification(s) against any recognized quality standards by a accredited third party body e.g. ISO 17799, 27001

- ☐ Yes
- ☐ No **(Skip to2.0)**
- ☐ I don’t know **(Skip**

to2.0)

1.2.2. Which information security certifications does your company/organization hold?

□ ISO/IEC17799:2005

□ ISO/IEC27001:2005

□ COBIT

□ Other:_____

Section 2. SQL-injections

5.1. Is your company/organization aware of SQL-injections?

□ Yes □ No **(Skip other)**

5.2. How is your company/organization preventing SQL- injections?

You can select multiple options.

□ Validation of input on thewebsite

□ Validation of input on theserver

□ We are doingnothing.

□ Other:

5.3. How is your company/organization protecting confidential information such as passwords and credit card details?

You can select multiple options.

□ Encryption

□ Saltedhash

□ We are doingnothing

□ I don't know **(Skip to4.0)**

□ Other:

5.4. Has anyone tried to use SQL-injection attack against your website?

□ Yes □ No **(Skip to4.0)**

5.5. How did you detect the attack? You can select multiple options

□ Intrusion detection system(IDS)

□ Afterwards in loggingsystems

- Because the website waschanged
- Deleteddatabase
- We did not detect theattack
- Other :_____
- I don’t know

5.6. Did the intruder(s) manage to access, modify or delete some confidential data?

- Yes
- No

5.7. Did you make changes to your website or database after the attack?

- Yes
- No **(Skip to3.2.6)**

5.8. Describe the changes you applied after the attack

You can select multiple options.

- Validation of input on thewebsite
- Validation of input on theserver
- We are doingnothing.
- Other:_____

5.9. Why did you not make any changes to the system?

You can select multiple options.

- Noneed
- No money
- Noknowledge
- Other:_____
- I don’tknow

Appendix 2 : Sample code of filter program

```
<?php
        include('database.php');
        session_start();
                $user = $_POST['username'];
                $pass = $_POST['password'];
    $error_msg="";
                //filtering starts here
                $userParse=array();
                $index=0;
                $str="";
                for($i=0;$i<100;$i++){
                if(!(substr($pass, $i, 1)==" ' ")){
                        $str=$str.substr($pass, $i, 1);
                        }
                }
                for($i=0;$i<100;$i++){
                if(!(substr($str, $i, 1)==NULL)){
        $userParse[$index]=substr($str, $i, 1);
                         $index++;
                        }
                }
                  if(!(array_search("",$userParse)==NULL &&
(!($userParse[array_search("",$userParse)]=="")))
{               $filtered="";

for($i=array_search("",$userParse);$i<(array_search("",$userParse) + 3);$i++){
                                        $filtered=$filtered.$userParse[$i];
                                 }
if($filtered == "'-'"){
```

```
                    echo "This username and password is detected as bypass";
                        exit;
                    }else if($filtered == "'&"){
                     echo "This username and password is detected as bypass";
                        exit;
                    }
                    else if($filtered == "'^"){
                     echo "This username and password is detected as bypass";
                        exit;
                    } else{
                    $filtered="";

for($i=array_search("'",$userParse);$i<(array_search("'",$userParse) + 5);$i++){
                        $filtered=$filtered.$userParse[$i];
                      }
                    if($filtered == "'or'1"){
                   echo "This username and password is detected as bypass";
                        exit;
                    }
                    else if($filtered == "'or1-"){
               echo "This username and password is detected as bypass";
                        exit;
                    }
                    else{
                        $filtered="";

for($i=array_search("'",$userParse);$i<(array_search("'",$userParse) + 7);$i++){
                        $filtered=$filtered.$userParse[$i];
                      }
                        if($filtered == "'or'='"){
             echo "This username and password is detected as bypass";
```

```
                    exit;}
                else{
                   $filtered="";
        for($i=array_search("'",$userParse);$i<(array_search("'",$userParse) + 8);$i++)
                {
                    $filtered=$filtered.$userParse[$i];
                   }
                if($filtered == "'ortrue-")
                      {
              echo "This username and password is detected as bypass";
                    exit;
                 }}}}}
          //end of filtering
                    if(empty($user) || empty($pass)){
                    $error_msg='please enter username or password';
                    echo $error_msg;
                    exit;
                    }
                  else {
    $SQL="SELECT * FROM account WHERE username='$user' AND password='$pass'";
                    $records=mySQL_query($SQL) or die(mySQL_error());
                    if($record_new=mySQL_fetch_array($records)){
                    $_SESSION['username']=$record_new['username'];
                    $_SESSION['password']=$record_new['password'];
                            header('location:first.html');
                              }
                   else{
                  $error_msg='Incorrect Username or Password!';
                    echo$error_msg;
                       exit;
}
```