CLASSIFICATION OF SQL INJECTION ATTACKS USING FUZZY TAINTING

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

In

Information Security

Submitted By

SURYA KHANNA

Roll No. 801433028

Under the supervision of

Dr. Anil Kumar Verma

Associate Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004, PUNJAB, INDIA

June 2016

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, *"Classification of SQL Injection Attacks Using Fuzzy Tainting"*, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Information Security* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Anil Kumar Verma and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Surya Khanna

801433028

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Dr. Anil Kumar Verma
Associate Professor
Computer Science and Engineering Department

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S.S. Bhatia)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGEMENT

Words can't express my gratitude towards my guide, **Dr. Anil Kumar Verma**, Associate Professor, Computer Science and Engineering Department, Thapar University, who has seen me through the entire tenure of the work presented in this thesis. I have been fortunate to have an advisor who gave me the freedom to explore the ocean of research on my own and at the same time the guided me through tumultuous waters. He is an inspiration in the truest sense of the world.

I am also thankful to Dr. Deepak Garg, Head of Department, CSED and Ms. Jhilik Bhattacharya, P.G. Coordinator, for their constant supervision of the thesis work. I would also like to thank my classmates who were always there to offer me, the help and facilities required for the successful completion of my thesis.

Most importantly, I would like to thanks my parents and the almighty for always steering towards the right direction, to help me remain imperturbable in the oddest of the times and take another step forward even when all hope was lost.

Surya Khanna
801433028
ME-IS
(2014-2016)

ABSTRACT

The interactive websites/applications involving database services need to assure the *confidentiality*, *integrity* and *availability* of data. They face a dire threat in the name of SQL Injection. A SQL injection attack is an act of manipulating the input data at the client end in order to perform an illegal operation on the database of targeted online system. They enable the attackers to obtain unrestricted access to potentially sensitive information. This may jeopardize the security of Web sites/applications, which may consequently lead to loss of users/customers trust. This report presents a SQL injection (SQLI) threat level indicator based on fuzzy logic for handling SQL injection attacks. The fuzzy tainting approach helped ruling out the possibilities of false positives.

***Keywords*— SQL injection, Fuzzy Logic, Tainting, Web Security, Threat level indicator**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

In recent times, the proliferation of web applications, as a means of information circulation and trading has empowered “data” as a crucial source to all business, banking endeavors and countless other activities. Any breach in data causes hindrance or total disruption of these activities making “Information Security” an essential feature in corporate ventures in order to maintain a competitive edge. The stakes involved in data loss are high and entail the potential for intellectual property theft, financial loss, compromising client information, public humiliation, ultimately jeopardizing organizations reputation.

1.1 MOTIVATION

Today’s web applications are constantly evolving, introducing new products and implementing better technologies to increase the efficiency of existing systems. They are often integrated with legacy systems. This heterogeneous nature of IT infrastructure has exposed more attack surface for aggressors to take advantage.

After doing online business and transactions for years now, organizations have become experts in covering up known loopholes and strengthening their defenses against cybercrime, although the statistics show a different picture. Fig. 1.1 shows the recent attack trends over the past years.

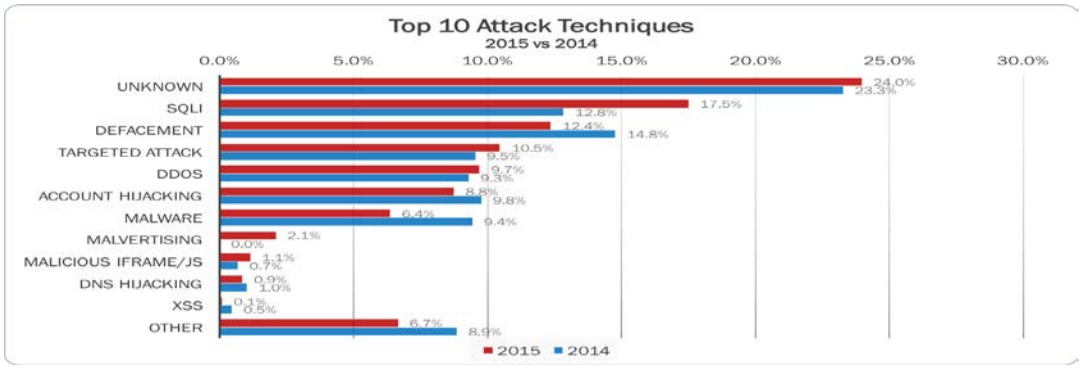

Fig. 1.1: Comparison between rising trends of attacks in past 2 years [1]

Attacks without any peculiar nature have dominated the field, threatening the web applications. These are the attacks that require a good amount of expertise at the end of security professionals, who mostly tackle them on the fly. However, over the past year, another attack, namely, SQL injection attack has become second to the topmost threat, topmost if we consider the known vulnerability attacks. This vulnerability has been encountered on and again by the web application, despite the numerous attempts to harden the systems against it. It is an active open ended topic of research with the research scholars.

This thesis focus on analyzing SQL injection attack and providing a solution to detect it, not only by filtering the points on input, instead looking at the complete query to figure out the harm it can cause on an online system.

1.2 DATABASE AND STRUCTURED QUERY LANGUAGE (SQL)

Database is an organized assortment of data in tabular format or objects. Database Management System (DBMS) is an interactive software system corresponding to the database that not only interacts with the users but also manages and analyzes the data stored in the database. It is also responsible for data transmission and exchange of data with other applications.[2].

SQL[3] is an ANSI standard language for conducting the database business. It provides the means to define, access, manipulate and process the data stored in the relational databases. It is based on theories of relational algebra and tuple relational calculus. The structure of SQL can be broadly categorized as:

a. Data Definition Language (DDL)

b. Data Manipulation Language (DML)

c. Data Control Language (DCL)

1.3 WEB APPLICATION SECURITY

Over the last 15 years or so, the web has become propitious and inexpensive channel for millions of businesses to communicate and exchange information coupled with financial transactions of clientele. Majority of the web applications[4]–[6] are laid on the basis of client-server architecture where the user feeds information to the client (web browser) which is grasped by the server for storage and processing. In regards to the technical aspect, the web is a tremendously engaging habitat of mass customization deploying a massive range of web applications at the beck and call of millions of global users. Fig. 1.2 delineates the basic working of a web application.

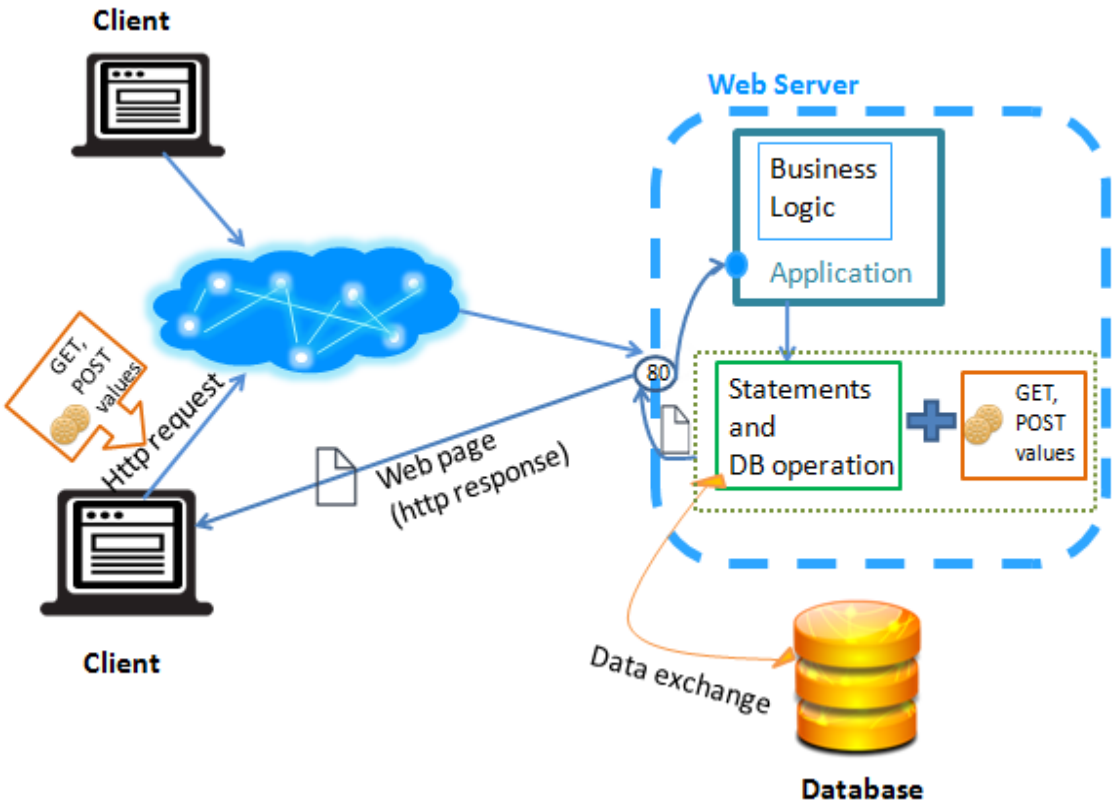

Fig. 1.2: Client - Server model

Although the websites has eased the way to carry on business, they have suffered various acts of vandalism and corporate defacement. Hackers have recently adopted the trend of gaining access to sensitive information, mainly because of the financial gains by selling

of the data. Most of the web sites now-a-days allow securing, managing, storing and relying of crucial user data (e.g. personal information, credit card details, social security numbers, emails, username, passwords etc.) for immediate or regular use. Such valuable information stored in the databases makes them a frequent target of attack.

1.4 TYPES OF WEB APPLICATION ATTACKS

Web applications encounter numerous attacks on daily basis. Following is the list of the most common attack forms on web applications:

1.4.1 SQL Injection (SQLi)

This is a top rated attack involving a slyly crafted input created by the attacker to expose the confidential information contained by the database. It occurs when SQL vulnerability is caused due to unfiltered user input to escape characters is accepted by the system or type check constraints are not implemented strongly, is exploited in an online system.

1.4.2 Defacement

As the term indicates, defacement of a website means distorting the appearance of the website either partially or fully by capturing the web server. There are numerous reasons for hackers to deface a website starting from mocking the website owner to political/religious motivations. It may be done by replacing the website entirely or injecting some images, pop-ups or text which were not a part of the original web page(s) or inserting malicious codes which can instill worms, viral codes in the visitor's computer thus making them reluctant to trust the website.

1.4.3 Distributed Denial of Service

Denial of service (DOS) is the method to renounce an authorized user's right to access a service. When multiple compromised systems aka botnets are influenced to inundate the resources of a targeted a victim server, then this process of unavailability of service is known as DDOS. In this attack form, the victim system is flooded with traffic from hundreds or thousands of botnets, so it becomes very difficult to distinguish between a legitimate and a fake request.

1.4.4 Hijacking

Hijacking means to take control of the actions or the work by the attacker. In Web application, hijacking is encountered in many forms, namely session hijacking, DNS hijacking, account hijacking.

a. Session hijacking: Taking control of cookies.
b. DNS hijacking: Taking control of DNS server and resolving legit names to attackers IP.
c. Account hijacking: Leakage of sensitive information like passwords, security answers.

1.4.5 Malware

Malware is a hypernym for any program that performs malicious tasks on a system or a network such as viruses, Trojans horses, worms etc. as directed by the attacker. They may act with stealth to spy on systems or cause an outright damage to disrupt the working of a system.

1.4.6 XSS Attacks

Cross side scripting (XSS) is vulnerability in a web application causing an indirect attack on the victim machine. In this malicious scripts are injected in legitimate websites which are executed on the victim's browser whenever a visit is made to the website infected with XSS vulnerability.

1.5 SQL INJECTION ATTACK

SQL injection is a weakness in a web application which is exploited by the aggressor by enclosing a malicious code in a legal input to a web application, which is in turn submitted to the SQL database and on execution, exposes the back-end database. The attacker covertly performs illegal activities such as unveiling confidential data, unauthorized access to data, illicit modification of data and eluding the authentication checks on the database. Organizations have to bear the burns of these undesirable events, harming its longevity and financial well-being.

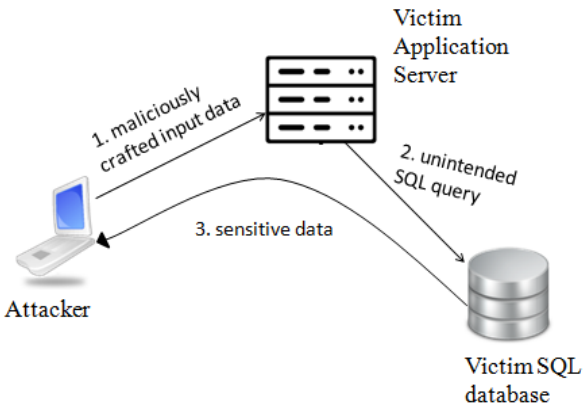

Fig 1.3: Basic SQL injection operation

1.6 FUZZY LOGIC

Fuzzy logic[7][8] is a mathematical means of enabling a computer to take decisions more in the terms of degrees of precision rather than analyzing everything as a the black and white picture. The shades of grey or the partial truths holds an important part of human decision making process and can sometimes change the entire perspective on things. Therefore, to lessen the prejudice we need to consider the in between possibilities to elicit a more realistic and accurate decision. Fuzzy logic is computationally defined as an approach based on “degree of truth” involving certain concepts or rules but vague boundaries.

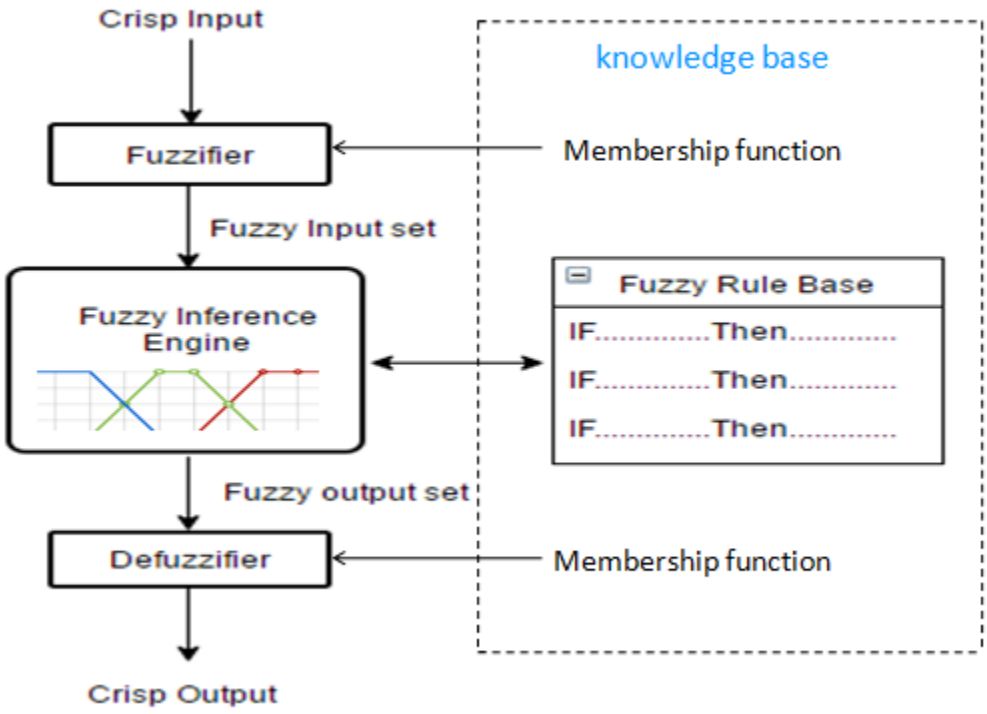

Fig. 1.4: A typical flowchart representing fuzzy decision making process

1.7 THESIS LAYOUT

The remainder of this thesis is presented as follows:

- CHAPTER 2: Explains SQL injection and the various ways through which it can be performed. It also briefs about the various research methodologies introduced to detect/ prevent SQL injection attacks. The gained knowledge helps to narrow down the peculiar problems that cause hindrance in plugging SQL injection attacks fully.
- CHAPTER 3: Illustrates the loopholes in the research work done to stop SQL injection and provides a brief overview about targets and techniques used to achieve them in this thesis.
- CHAPTER 4: Describes the environment used to implement fuzzy tainting. It also delineates the SQL injection attacks simulated and a brief overview of terminologies used in the thesis.

- CHAPTER 5: Provides the implementation details and the researched algorithm used to achieve the targets listed in chapter 3.
- CHAPTER 6: Concludes the thesis.

CHAPTER 2

LITERATURE SURVEY

This chapter gives the details about SQL injection and the work done till date to mitigate this attack. It also entails the need to explore new methods with a view to stop SQL injection attacks fully.

2.1 SQL INJECTION ATTACKS

The term “SQL injection” originated in 1998, although it made its debut in public in the year 2000 [9] and hasn’t disappeared till date. It is performed when the attacker changes the semantics of an input to the web application by inserting SQL commands to generate a result or affect the database in an undesirable manner. SQL injection attacks are executed by the attackers with the intent to imitate legal identities, sabotage the data, create invalidation issues like nullification of transactions, thoroughly reveal the database, make the data unattainable or obtain highest level of access to the database server.

The web application knows not the difference between the genuine and forged queries and responds by producing an output as dictated by the query, thereby supplying the attacker with unauthorized information. As a case, Asaf Orpani[10], a Trustwave SpiderLabs researcher, recently uncovered a SQL injection vulnerability hidden in Joomla (version 3.2-3.4.4), an open-source Content Management System (CMS) used by approx. 2.8 million websites. Session IDs of currently logged in users were extracted and even administrator access to one of the websites was obtained by the attackers. In October 2015, SQL injection was suspected for contribution behind the attack on the British telecommunications company Talk Talk's servers[11], which lead to the compromise of personal information of its approximately 4 million customers. In another case, the personal details of approximately 5,000,000 parents and greater than 200,000 kids was leaked in November 2015, after a Chinese company VTech Toys[12] was compromised by the hackers.

SQL injection tops the statistics for the most threatening attacks on online systems as in rank lists of organizations like OWASP[13], MITRE[14]. The most valuable asset of an organization lies amid danger as shown in the reports published by Imperva[15] and Verizon[16]. The Data Breach Investigations Report (DBIR) given by Verizon in 2015 enlists 2,122 confirmed data breaches out of 70 organizations.A typical application suffered 3 times more SQL injection attacks (SQLIA's) according to the facts compiled by Imperva's web application attack report edition #6[15]. Even alarming observation was an increase of 29.63% in SQLIA than the previous year.

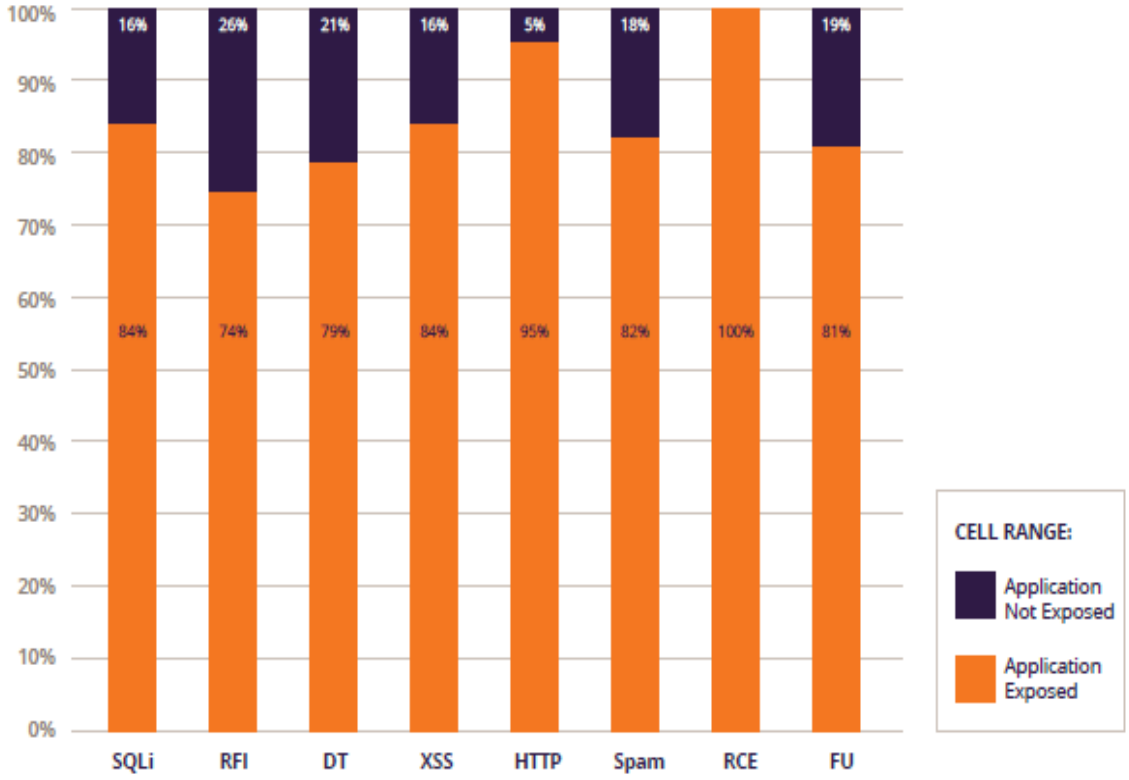

Fig. 2.1: Web application malicious traffic exposure ratio[15]

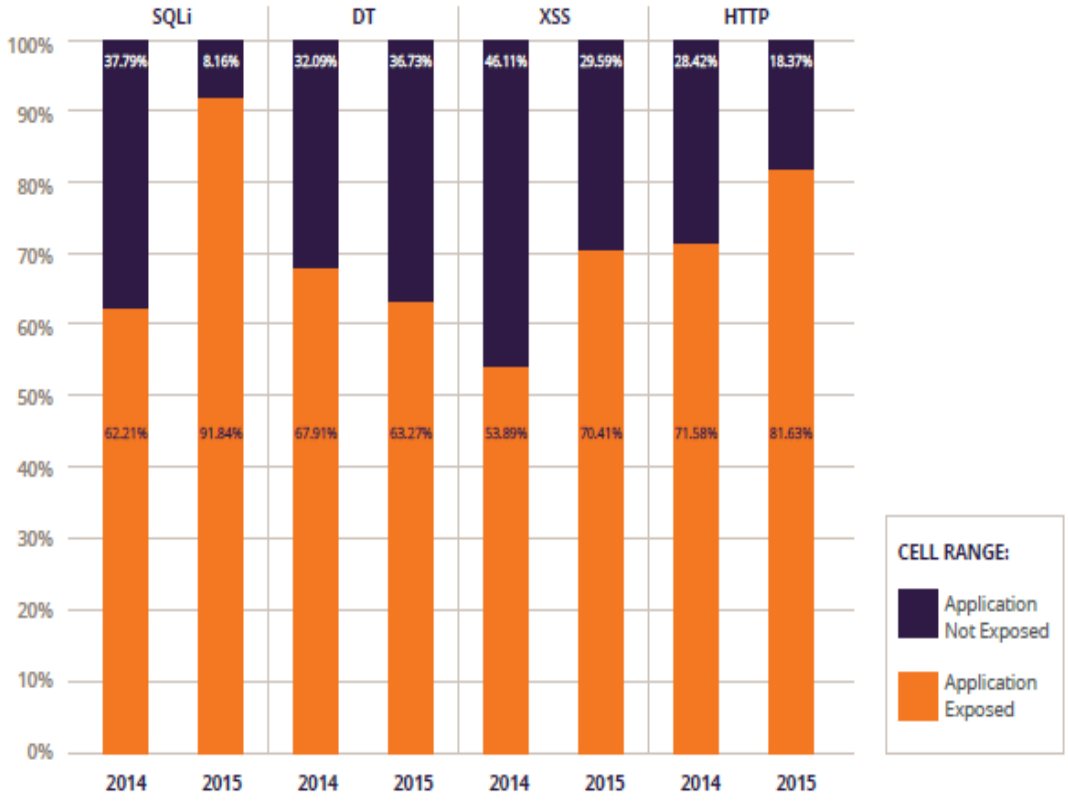

Fig. 2.2: Web application malicious traffic exposure ratio between years[15]

Further, Trustwave's Global Security Report 2015[17] enlists SQL injection as the third most prevalent web application vulnerability compromising the database, taking an unacceptable advantage of administrator's credentials, plundering intellectual properties and payment systems. It might be the third most exploited attack but nonetheless dangerous causing serious problems even after 17 years of its first discussion in 1998. In 2014, Out of 574 data compromises investigated by Trustwave SpiderLabs in various industries across 15 countries, SQL injection contributed to 15% of the total. An increase of 10% was observed in the frequency of SQL vulnerability in 2014.

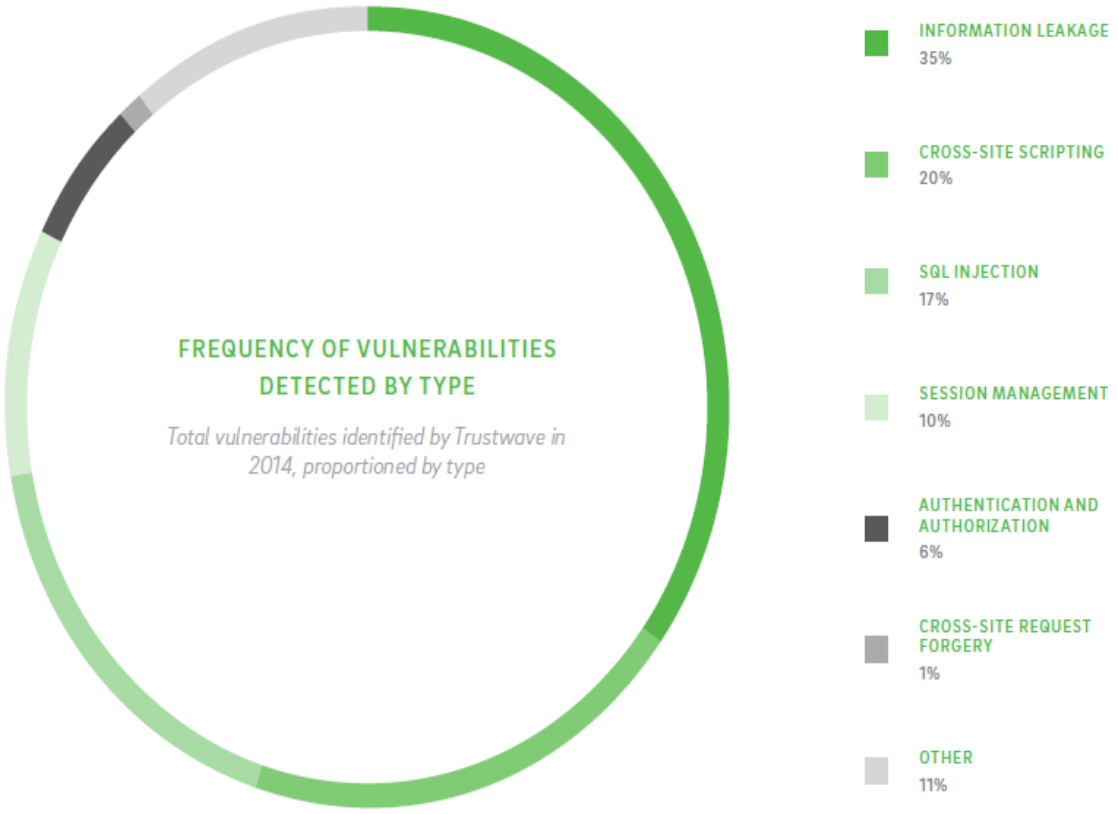

Fig. 2.3: Frequency of vulnerabilities detected [17]

2.2 TYPES OF SQL INJECTION ATTACKS

This section provides a brief background on SQL injection. Input sources for SQLIA's in online systems can be broadly categorized as:

- First Order/ Direct Attacks- In this type, the malicious code executes along with the SQL Query execution.
    - Through user input: In web applications, data is usually submitted to the business logic by forms via GET/POST HTTP requests. A maleficent user might input some illegal SQL queries from these entry points.
    - Through cookies: Cookies[18] are small files containing moderate amounts of user specific data sent by a particular website to the client side or the user's browser. They are read by the server whenever the user returns to the same website for authentication, tracking, maintaining state over stateless HTTP and maintaining user specific information such as

their site preferences, online shopping cart contents etc. If the contents of cookie are used to create SQL queries, the attacker can temper with the values of parameters contained in the cookies and execute malevolent activities.

- Via server variables: Server variables is a group of data obtained from the HTTP or network headers and the environment variables. The data can be altered so it is advised to sanitize them before logging any such information in the database. Attackers can place a malformed SQL query directly into the header values which gets executed while we are trying to log the server variables.

- Second Order/Indirect Attacks- These are the attacks in which the malicious input is inserted at a place different from where the attack is intended to be performed i.e. the malicious code is executed at some later stage. No immediate effect is seen on the system nor do any results become readily available to the attacker.

SQL injection attacks are of following types[19]:

2.2.1 Tautology

The idea is to insert the specially crafted SQLIA into one or more conditional statements (usually in WHERE clause of queries) such that the condition equates true in every possible interpretation. The resultant query may lead to breach in confidentiality of data by bypassing authentication or displaying sensitive data to the attacker or the world. Example: Allergy records and complaints of the patients in a hospital can be retrieved by the attacker as given by:

Select patient_name, complaints, allergies from patients where patient id = ‘’ *or 1=1--* and passCheck = ‘’

This gravely endangers the security of a patient’s details, ultimately risking his life.

2.2.2 UNION

In this technique, an attacker can exploits a jeopardized parameter to maneuver the application into returning data from a table contrary to expected by the programmer. In turn, the attackers acquire total control on the second table, so it can be used to retrieve any information the second table stands to offer. Generally, this type of attack is used in SQL query to gain more than intended information by the attacker. Example: Attacker can retrieve credit card details from a banking application although the developer intended to give away only the account no.

Select name, id from user_details where ulogin= ‘’ *union Select cno, pin from credit_card* limit 1,1 -- uPassword = ‘’

2.2.3 Logically Incorrect Queries

The attacker intends to gather information about the database such as its type, structure or any metadata about the schema or any information about the server by causing syntax, type conversion or logical errors in the queries. These errors can help the attacker to deduce vulnerable parameters, extract data or the table names and perform database fingerprinting. *Example*: Attacker can cause a type conversion error to acquire relevant data.

SELECT ccNo FROM credit_cards WHERE pin=‘*’ convert (int,‘xyz’) -- ’*

2.2.4 Piggy Backed Queries

The general idea is to associate a malicious query to a perfectly valid and complete query. The original query remains untouched but it is accompanied by attacker’s malicious query. Therefore, the database executes multiple queries rather than a single query as intended by the developer. This sort of attack may render the CIA goals useless. The attacker can extract, modify, delete or perform any sort of operations on the tables in the database. Example:

SELECT * FROM students WHERE sLogin= ‘admin’ AND sPassword=’adm@134’; UPDATE Grades SET grade='A' WHERE studentId=56983;-- ‘

2.2.5 Inference

This form of SQLI attack is used to elicit information about a secured website by observing the behavioral changes in the function and responses from the websites. This is done by using two well known techniques:

a) *Blind injection* - *Information* is extracted based on the indirect true/false responses given by the application. Example: The attacker asks indirect questions by issuing queries. If both the queries responds with error messages then the login has been secured.

SELECT iUID FROM user_details WHERE uLogin='xyz*' and 1=0 --* ' AND uPass=''
SELECT iUID FROM user_details WHERE uLogin='xyz*' and 1=1 --* ' AND uPass=''

b) *Timming attack* – In this the delays or the time values are observed to deduce any information. Example: In this the attacker is trying to determine the table name by asking if the 1st character of the table name has ASCII value greater than Y. If so, then cause a delay of 5 seconds.

SELECT acc_no FROM accounts WHERE accLogin='admin' and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > Y WAITFOR 2 -- ' AND accPass=''

2.2.6 Stored Procedures

In this form of attack, the attacker manipulates the parameters passed to stored procedures. Not only can the attacker insert malicious codes into procedure but the stored procedure itself is vulnerable to attacks like buffer overflow. Here is an example showing the attacker causing system shutdown by passing ';shutdown;-- in any of the parameters (loginId, password) of the following stored procedure.

CREATE DEFINER=`root`@`localhost` PROCEDURE `userDetailProc`(IN loginId varchar(45), IN password varchar(45))

BEGIN

Select iUID , sFirstName, sLastName, sEmail, cGender from user_master

where login = loginId and pass = ;

END

2.2.7 Alternate Encodings

This technique is used in parallel to other SQLIA's in order to evade detection. The injected query is encoded say in ASCII format to prevent detection. Example: While passing the shutdown command in the procedure mentioned above, we encode it and pass it as

'; exec(0x73687574646f776e) - -

2.3 LITERARTURE SURVEY

Over the years, various methodologies that have been proposed by researchers to evade SQLIA are discussed in this section.

In 2003, Huang et al. [20] proposed a black box approach to assess the security of web application using fault injection and behavior monitoring. It uses a web crawler to determine SQLIA target points of a web application. A knowledge expansion model is used to learn the behavior of malicious pattern and determine high confidence terms. Lastly, the negative response extraction (NRE) algorithm is used to determine the impact of input on the results.

Later on, Stephen W. Boyd and Angelos D. Keromytis,2004 [21] used a secret key to randomize the SQL queries with the help of a proxy. The security of this technique depends on the strength of the secret key. In another approach, an information flow model [22] known as Web application Security by Static Analysis and Runtime Inspection given by Yao-Wen Huang et al. is used to express the trust level on variables. In this, AST (Abstract Syntax Tree) is generated using lexer and parser. Verification is

performed by type state tracking to perceive insecure information flow and correspondingly inserts a sanitization routine.

In 2005, William G.J. Halfond and Alessandro Orso [23] built SQL query models combining static as well as dynamic monitoring techniques and created a tool named AMNESIA(Analysis and Monitoring for Neutralizing SQL Injection Attacks) to detect injection. In this, rigorous scanning of web application is done to identify the hotspots for issuing SQL queries in the analysis module. These hotspots are then checked in a SQL query model which is a non-deterministic finite state automaton to determine any violation in all possible query cases that can be issued for a hotspot.

In 2006, Yonghee Shin, Laurie Williams and Tao Xie [24] combined static analysis, runtime detection and automatic testing producing an automated prototype tool named SQLUnitGen to determine the existence or the efficacy of black or white list input filters in a web application. It used AMNESIA to trace the input flow. The test cases are generated and refined with attack input to locate hotspots. It ultimately outputs the secure and vulnerable methods in the form of colored call graph.

Further, X. Fu, et al. in 2007 [9] proposed a White box model for static analysis of byte code to determine vulnerabilities at compile time. For every hotspot, a monitoring function consults an attack library containing a set of pre-determined attack patterns, based on which hybrid string constraints are generated. A constraint solver is then used to determine any breaches in security by determining satisfaction of path constraints.

K. Kemalis and T. Tzouramanis [25] utilized security specifications used by the programmers to write down the SQL queries and performed lexical and syntactical analysis on the web application code to prevent all forms of SQLIA’s from executing on the database in the year 2008. Whereas William G.J. Halfond et al. [26], in the same year, proposed a novel approach based on positive tainting and the concept of syntax-aware evaluation implemented in the form of Web Application SQL-injection Preventer (WASP) tool to stop all of the otherwise successful attacks.

In 2009, work was done on stimulating the attacks by Yang Haixia and Nan Zhihong to harden the web applications against SQLIA’s. In this scheme, they used breadth first

method to scan the web application for determining the entry points. On top of that, they used an attack rule library to generate test cases and launch attack on websites, to determine the flaws in the system.

Further, in 2010, Bisht et al brought forward CANDID [27], a query-structure mining approach to determine the deviation of queries formed by candidate input from programmer intended queries. The next year, 2011, Sangita Roy, Avinash Kumar Singh and Ashok Singh Sairam proposed a light-weight and fast SQL injection vulnerability scanner[28] based on filtering HTTP request sent by clients and compare with attack signatures to discover the vulnerabilities and determine the effectiveness of counter attack mechanisms in a web application.

In 2013, A.S. Gadgikar [29] came forward with a negative tainting approach to detect SQLIA. In this approach all entry points are checked for known attack keywords and later on Dennis Appelt et al,2014 [30] worked to automate the black box testing of SQL injection Attacks. In this approach, test cases are generated using a set of mutation operators that manipulate inputs to create new test inputs triggering SQLiA. A prototype of mutation operators was proposed to create ambiguous SQL injection code in order to bypass security filters, such as web application firewalls, undetected. In another approach, A. Sadeghian et al [31] sanitized the input variables in the headers of HTTP request (GET/POST) by using escape characters to prevent SQL injection.

Recently, in 2015, B. Hanmanthu and colleagues [32] proposed data mining technique using decision trees to classify attack signatures thereby preventing SQL injection. Decision making is based on associative classification rules.

2.4 COMPARISON

Table I shows the comparison between various methodologies in regard to their defensive abilities against various types of SQLIA’s.

TABLE I: Comparison between different schemes to stop different types of SQLIA's

Attack Types / Scheme	Tautology	Illegal Queries	Union	Stored Procedure	Piggy Backed Queries	Inference	Alternate Encoding
WAVES	*	*	*	*	*	*	*
SQLrand	✓	✗	✓	✗	✓	✓	✗
AMNESIA	✓	✓	✓	✗	✓	✓	✓
CSSE	✓	✓	✓	✗	✓	✓	✗
IDS	*	*	*	*	*	*	*
SQLCheck	✓	✓	✓	✗	✓	✓	✓
SQLGuard	✓	✓	✓	✗	✓	✓	✓
SAFELI	✗	✓	✓	✓	✓	✓	✓
CANDID	✓	✗	✗	✗	✗	✗	✗
WebSSARI	✓	✓	✓	✓	✓	✓	✓
WASP	✓	✓	✓	✓	✓	✓	✓

The symbol ✓ means that the method successfully handles that type of SQL injection attack while the symbol ✗ indicates that it cannot stop the respective type of attack. There is another symbol '*' used in the above table which indicates partially addressing of the corresponding attack.

CHAPTER 3

PROBLEM STATEMENT

3.1 GAPS IN STUDY

Most web applications have laid down strong defenses against SQLIA's and have achieved a notable amount of success in preventing them. Although, these defense schemes sometimes come too strongly and treat valid inputs unjustly. Restrictions are put on the type of characters that can be entered in an input box or special characters are escaped before transmission.

Even with the numerous automated techniques proposed for prevention of SQLIA, they fail to understand the semantics of the queried statements, ultimately failing the fine line of separation between legitimate inputs and the malicious ones.

3.2 AIMS AND OBJECTIVES

This thesis is written to propose a novel approach to counter SQLIA by understanding the meaning of an overall SQL query from a rationale mind's perspective using an approach which brings us closest to the way a human thinks commonly known as Fuzzy Logic. The main objectives are listed as under:

- Simulation of SQL injection attacks
- Classification of risk level engendered by SQLIA
- Thwarting of SQLIA

3.3 METHODOLOGY

- Creating a dummy web application using JAVA and MySQL
- Applying Fuzzy Logic using jFuzzyLogic[33][34] library to prevent SQL injection
- Using JFuzzyChart for graphical representation for classification of threat imposed by SQL injection attack

CHAPTER 4

ENVIRONMENT

To perform this experiment, a dummy website in struts framework (JAVA) was created with least cautions taken in regards of security against SQL injection attacks. It contained links to execute various types of SQL statements, to facilitate the experimentation. The following images show the dummy website and simulation of various SQL injection attack types. The backend database used for this dummy website is MySQL.

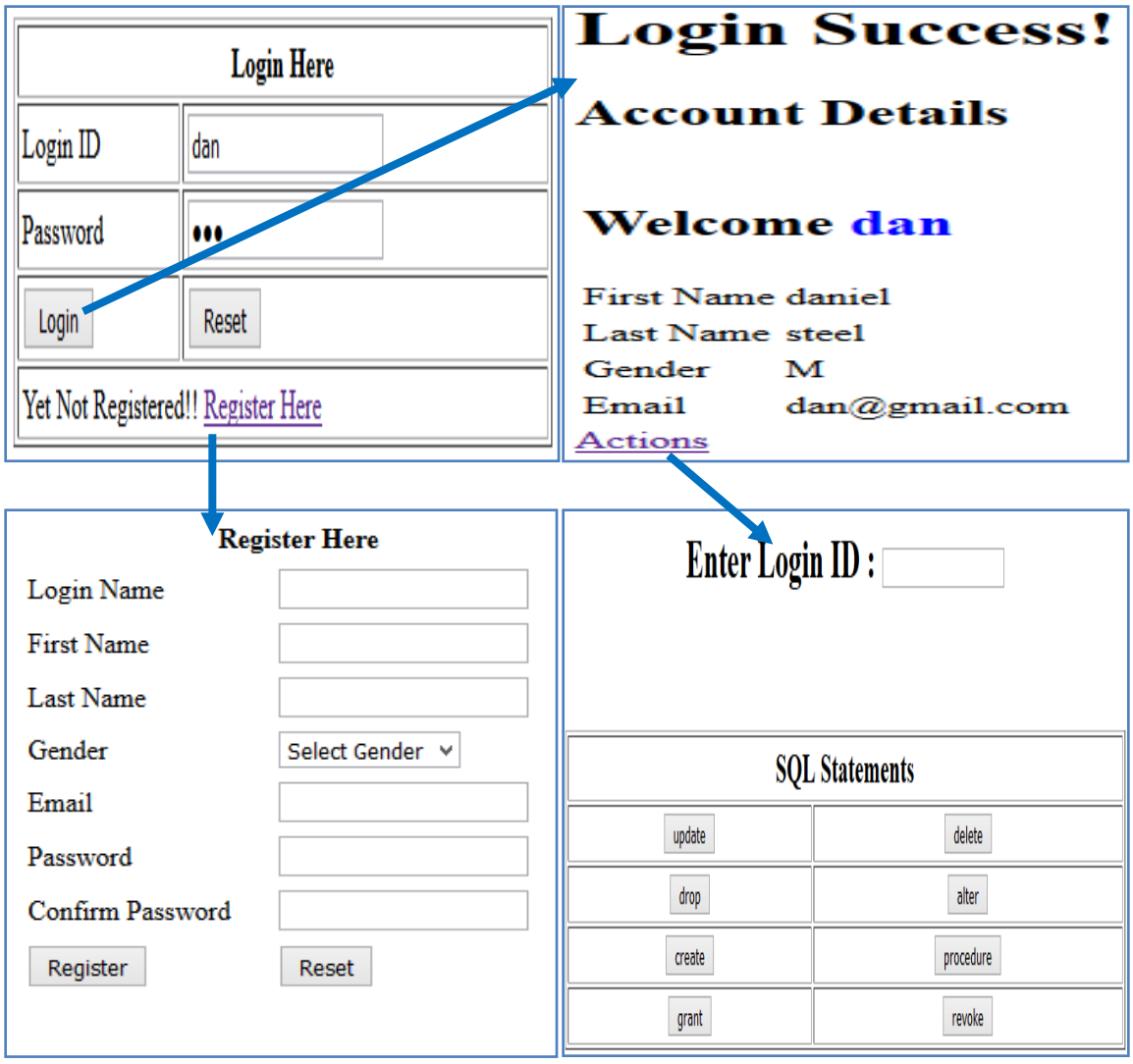

Fig. 4.1: Dummy website vulnerable to SQL injection

4.1 SIMULATION OF SQL INJECTION

Various types of SQL injection attacks were simulated on the dummy website, as shown through following examples:

1. *Tautology Attack*

Enter " ' or 1=1--" in Login ID field of login page

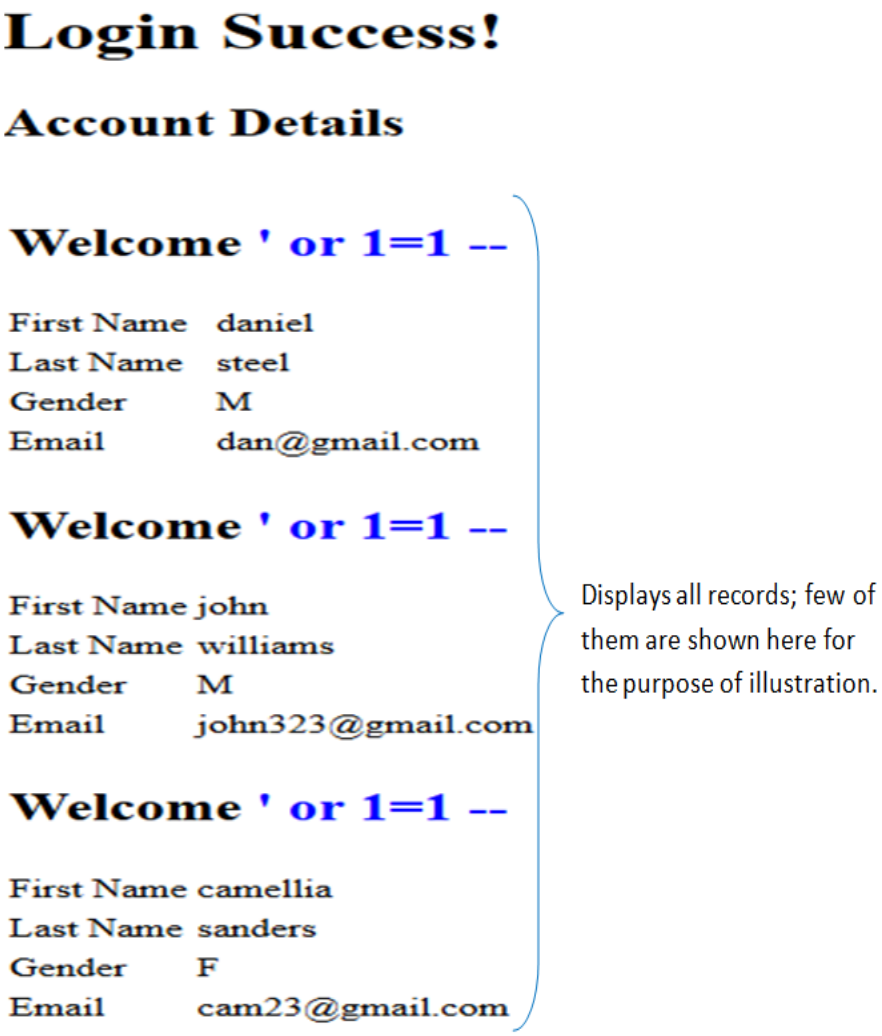

Fig. 4.2: Tautology Attack

2. *Union Attack*

Enter "' union Select sFirstName, sLastName, sEmail, cGender from user_master where sFirstName = 'daniel' --" in Login ID field of login page

Fig. 4.3: Union Attack

3. *Illegal Query Attack* (Type conversion error generation for fingerprinting)

Enter "convert (int,(select top 1 name from sysobjects where xtype='u')) " in Login ID field of login page.

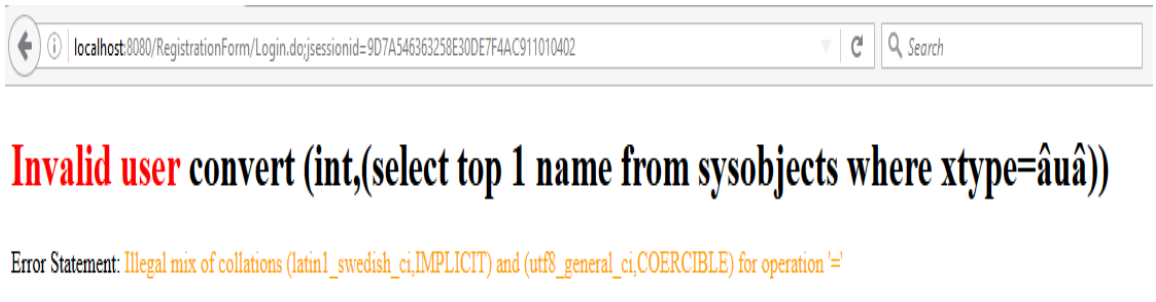

Fig. 4.4: Illegal Query Attack - Type conversion error

4. *Illegal Query Attack* (Syntax error generation for fingerprinting)

Enter "' orderby asc --" in Login ID field of login page.

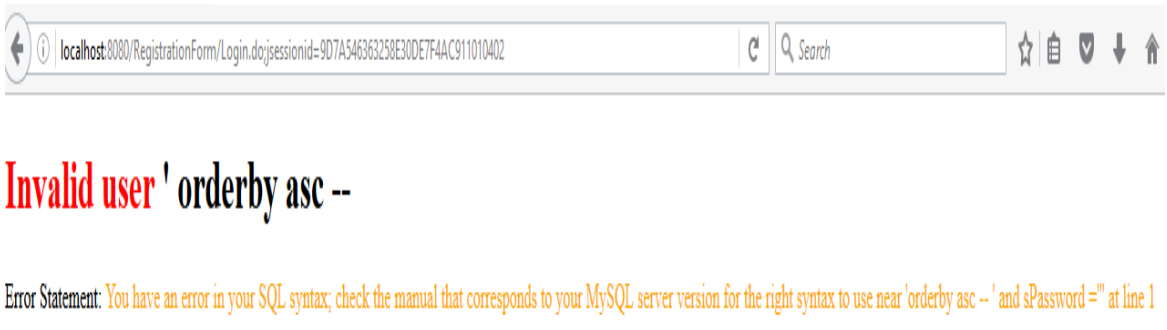

Fig. 4.5: Illegal Query Attack - Syntax error

5. *Piggy-backed Query Attack*

22	robert	steel	robert@gmail.com	robert	robert	M	2016-05-28 15:27:51
23	chandler	bing	chandler3@gmail.com	bing	bing	M	2016-05-28 15:27:51
24	perry	williams	perry7@gmail.com	perry	perry	M	2016-05-28 15:27:51
25	andy	clinton	andy@gmail.com	andy	andy	M	2016-05-28 15:27:51
26	nina	sharma	nina@gmail.com	nina	nina	F	2016-05-28 15:31:54
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig. 4.6: State of database before injection

Put "');UPDATE user_master SET login='malicious' WHERE sEmail = 'nina@gmail.com' --" in the login field of registration page.

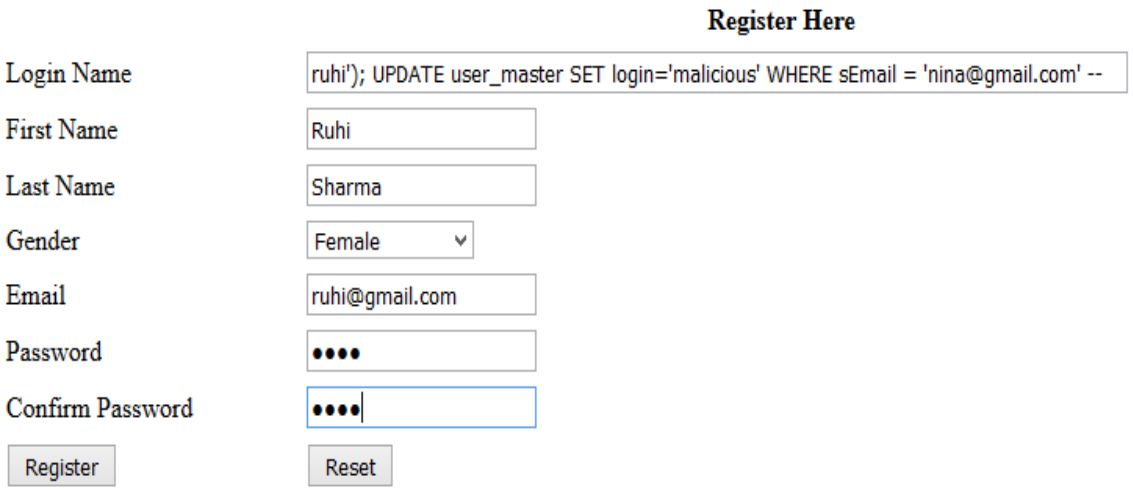

Fig. 4.7: Piggy-backed Query Attack

iUID	sFirstName	sLastName	sEmail	login	sPassword	cGender	sCreatedDate
22	robert	steel	robert@gmail.com	robert	robert	M	2016-05-28 15:27:51
23	chandler	bing	chandler3@gmail.com	bing	bing	M	2016-05-28 15:27:51
24	perry	williams	perry7@gmail.com	perry	perry	M	2016-05-28 15:27:51
25	andy	clinton	andy@gmail.com	andy	andy	M	2016-05-28 15:27:51
26	nina	sharma	nina@gmail.com	malicious	nina	F	2016-05-28 15:31:54
31	Ruhi	Sharma	ruhi@gmail.com	ruhi	ruhi	F	2016-05-28 18:01:53
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig. 4.8: State of database after injection

6. *Inference Attack* (blind injection)

a) Input xyz' and 1=0 -- in Login ID field of login page.

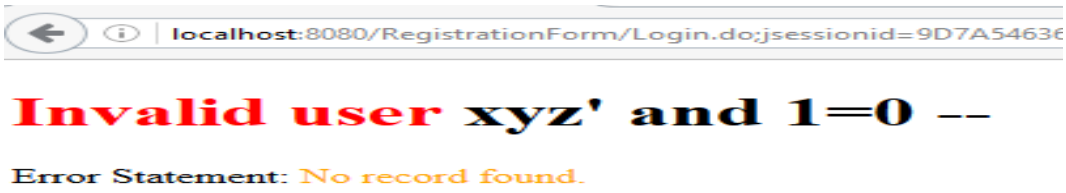

Fig. 4.9: Blind Injection Attack part (a)

b) Input: dan' and 1=1 -- in Login ID field of login page.

Fig. 4.10: Blind Injection Attack part (b)

7. *Inference Attack* (Timing Attack)

Input: xyz' – IF (MID(VERSION(),1,1) = '5', SLEEP(15), 0) LIMIT 0, 1 -- in Login ID field of login page.

Output with delay >15sec

Fig. 4.11: Timing Attack

8. Attacking Stored Procedures

Enter "dan'); shutdown; --" in the login Id field of action page.

Fig. 4.12: Attacking Stored Procedures part (a)

Later on when any other command/procedure is attempted to be executed, we are unsuccessful as the SQL server is not running.

Fig. 4.13: Attacking Stored Procedures part (b)

This is also an example of second order attack as when the malicious query is entered the expected operation is performed but as soon as any other query is made to the database, no response comes because the connection to the database has been terminated.

4.2 FUZZY LOGIC

The information perceived by organs of human body is often ambiguous and incomplete. The human brain is programmed to interpret any signals, erroneous or not, in a manner which makes sense. Fuzzy set theory provides the required vagueness to the classical logic methods in order to interpret indistinct information with the help of membership functions. Following is a brief overview of various terminologies used in fuzzy logic:

4.2.1 Fuzzy Logic Control

The theoretical concepts of fuzzy logic are implemented for industrial control automation in the form of Fuzzy Logic Control (FLC's)[35] [36]. FLC's are linguistic rule based models implemented to automate the control strategies formed by expert knowledge. The strong approximation features as well as the ability to provide humane experience make FLC's well favored for applications that lack in achieving good results through classical approaches. Fuzzy control implements fuzzy logic so it has the ability to form multiple values, thereby it is no longer restricted to the "true"/ "false" values of control

propositions. Hence, Fuzzy control systems are desirable for control actions based on a given set of inputs.

4.2.2 Membership Functions

Membership[7] as the word indicates means being a part of, you can be a member or not but when related to fuzzy set theory this precise membership definition changes to "degree of membership" i.e. whether anything is partially a part of something or not? It is defined in the mathematical/numerical terms in FLC by membership functions and is represented by μ.

4.2.3 Fuzzification

Fuzzification is the process of transformation of crisp inputs to fuzzy inputs with the help of membership function. It is mathematically stated as:

$$\tilde{A} = \mu(\boldsymbol{x})$$

where μ is the membership function.

4.2.4 Fuzzy Inference

The fuzzy inference engine used in this work is based on an inferring procedure known as Generalized Modus Ponens (GMP). GMP[8] is mathematically stated as:

IF x is A then y is B

x is A'

---

y is B'

where A and B are fuzzy sets.

4.2.5 DeFuzzification

DeFuzzification[7][8] is the process of transformation of fuzzy inputs to crisp inputs with the help of membership function. This work use a very popular defuzzification method

known as Center of Gravity (COG) which works by obtaining the center of the total area occupied by the fuzzy set. It is mathematically stated by the formula:

$$x^* = \frac{\int \mu(x)\, x\, dx}{\int \mu(x)\, dx}$$

CHAPTER 5

IMPLEMENTATION

This thesis proposes a novel approach to mitigate SQL Injection by fuzzy tainting with the target to minimize false positives. This research work emphasizes on detecting the attacks as deemed closest to the human perception.

5.1 PROPOSED SCHEME

Our technique relies on inspecting the overall impact of running a query rather than pinpointing individual symbols or keywords deemed dangerous. Fuzzy logic helps create vagueness in crisp boundaries, thereby allowing us to validate legitimate inputs, otherwise considered malicious.

5.2.1 Overview

In this technique, we initially use negative tainting [29] to determine any malicious string in input parameters, attempt to determine their attack types according to which we can associate a severity level with them as shown in table II. The Risk level is based on the hindrance, it can cause in the smooth working of a system. Scores are calculated from the taints found against the known attacks for all input parameters of a query. These scores are used to indicate the risk level (as shown in fig) of the malicious input strings.

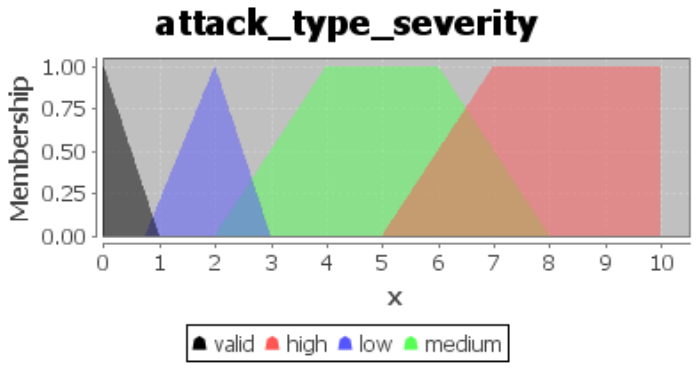

Fig. 5.1: Input threat level

Attackers use logically incorrect queries or inference techniques to determine information about the database though error messages. Not much information can be retrieved through these methods and are usually considered as a part of database fingerprinting. So, no sensitive data breach happens, thereby they are enlisted to pose a low level threat. Whereas attacks like tautology, union, stored procedure may lead to breach of confidentiality and hence, given a relatively higher level threat indication.

Table II: Severity level of different types of SQLIA's

MODE OF ATTACK	EXAMPLE	RISK LEVEL
Tautology	' or a=a --	Medium
Logically Incorrect Queries	Pass character in integer data type e.g. pin = '#123\' or pin= convert (int,'xyz')	Low
Union Query	' UNION SELECT accNo, pin from user_detail where iUID = 1349 --	Medium
Piggybacked Queries	I)'; insert into users values(666, 'attacker', 'admin', 0xffff)— II)';Drop table accounts;--	High
Stored Procedures	Pass following as parameters I) '; SHUTDOWN; -- II)';Drop table users;--	Medium, High
Alternate Encoding	'; exec(char(0x73687574646f776e)) --	Medium, High
Inference **I)Blind Injection** **II)Timing Attacks**	 I) john' and 1=0 -- john' and 1=1-- II) id=1') or sleep(25)=0 limit 1--	Low

Then we look over the programmer intended query corresponding to the input parameter and determine the probability with which our system will be affected if any malicious input is given to the query.

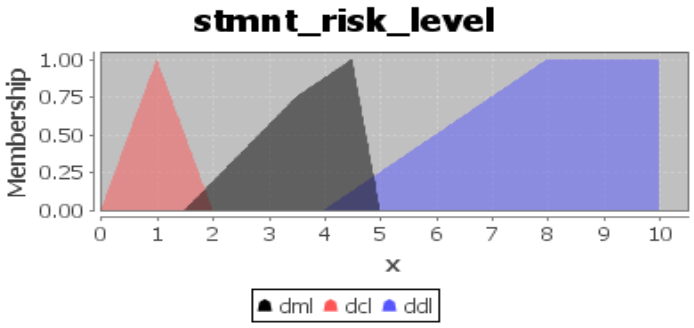

Fig. 5.2: Impact of Statements to system under attack

Malicious content in a DDL poses highest level of threat. Any modification at the schema level can lead to huge loss/leakage of sensitive data as they can affect the entire table, e.g. the attacker can create another table in which he/she can dump sensitive information about the system and make it available for access by unauthorized users or a table containing monetary records can be deleted affecting large no. of users. While DML statements are considered more dangerous than DCL. Any malicious content in DML statements can lead to breach in confidentiality or integrity of data.

TABLE III: Risk imposed on application according to type of SQL Query under attack

STATEMENT TYPE	STATEMENT	IMPACT
DCL	Revoke	Low
DCL	Grant	Average
DML	Select	Average
DML	Insert	Average – High
DML	Update	High
DML	Delete	High
DDL	Create	Average
DDL	Alter	High
DDL	Drop	High
DDL	Truncate	High

Fuzzy logic basically reduces paradoxes or multi-valued logic to half-truths (or half-falsities). It is implemented to solve such half truth/ half false cases encountered while detecting malicious strings causing SQL injection attacks. A Quote (‘) is considered as a dangerous parameter for SQL inputs, although it is partially true. We encountered places where the quote is a part of a perfectly valid input. Due to the various guidelines laid down in the defense strategies of SQLIA’s, these legal inputs are also considered dangerous. To resolve such problems, we have designed a set of fuzzy rules in their deductive form. A fuzzy rule set used to determine the overall threat level.

For each query, the set of rules is applied to obtain the degree of membership value which is then defuzzified to produce the output. To predict the exposure of query to SQL injection attacks, we use Center of Gravity (COG) method of defuzzification [7] given by the algebraic expression:

$$x^* = \frac{\int \mu(x)\, x\, dx}{\int \mu(x)\, dx}$$

5.2.2 ALGORITHM

To determine the threat imposed by an input, not only the input is analyzed in fuzzy logic but we also take under consideration the impact caused by its insertion according to the nature of targeted statement under attack. The following decision making tree delineates the detection process graphically.

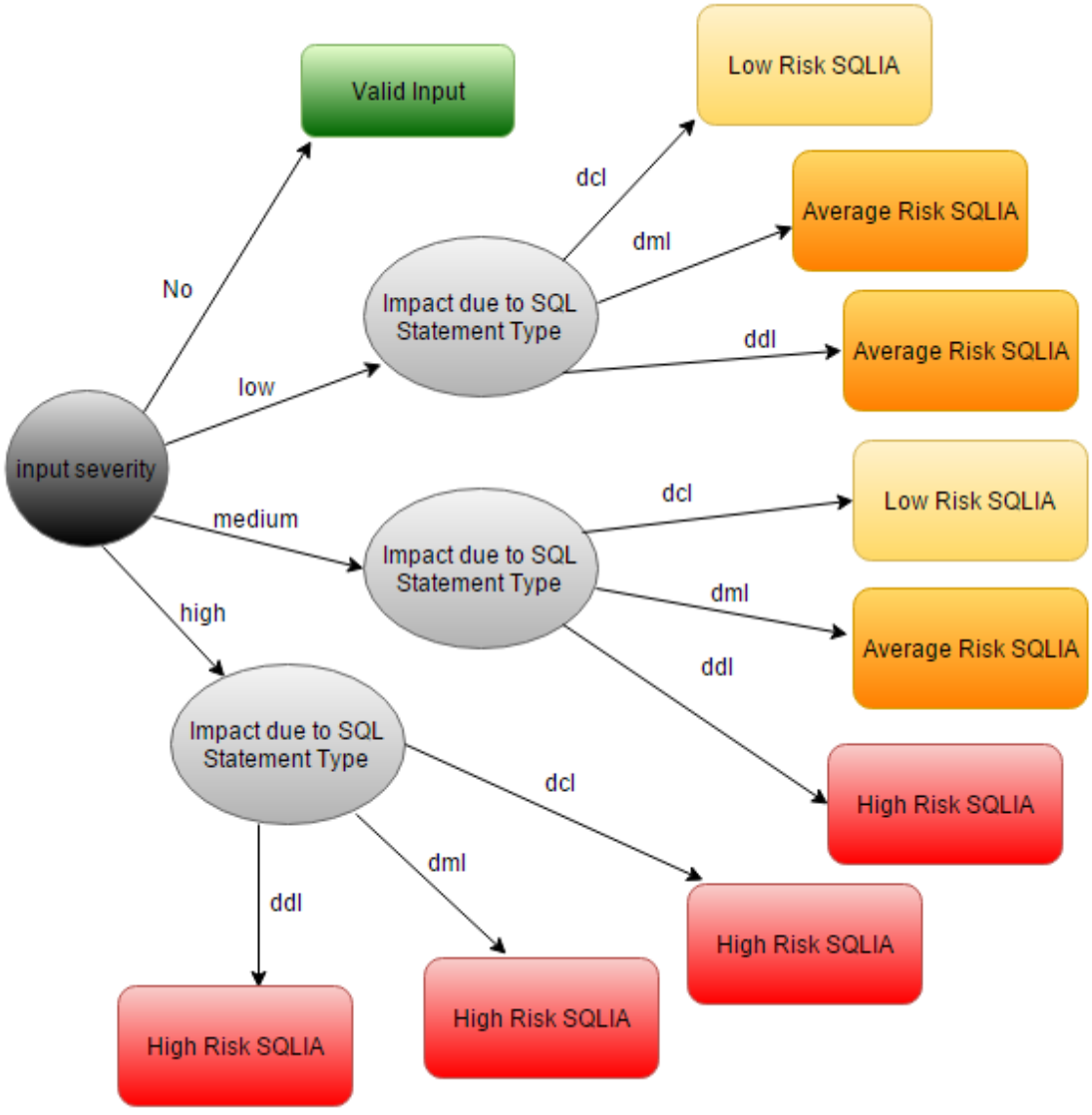

Fig. 5.3: Decision making process for SQLIA detection

The basic fuzzy logic rule base algorithm used in implementation of fuzzy tainting is given By Fig 5.4:

```
RULE 1 : IF attack_type_severity IS valid THEN sqlia IS no_risk;

RULE 2 : IF attack_type_severity IS low AND stmnt_risk_level  IS dcl
            THEN sqlia IS low;

RULE 3 : IF attack_type_severity IS medium AND stmnt_risk_level  IS dcl
            THEN sqlia IS low;

RULE 4 : IF attack_type_severity IS high AND stmnt_risk_level  IS dcl
            THEN sqlia IS severe;

RULE 5 : IF attack_type_severity IS low AND stmnt_risk_level  IS dml
            THEN sqlia IS low;

RULE 6 : IF attack_type_severity IS medium AND stmnt_risk_level  IS dml
            THEN sqlia IS average;

RULE 7 : IF attack_type_severity IS high AND stmnt_risk_level  IS dml
            THEN sqlia IS severe;

RULE 8 : IF attack_type_severity IS low AND stmnt_risk_level  IS ddl
            THEN sqlia IS average;

RULE 9 : IF attack_type_severity IS medium AND stmnt_risk_level  IS ddl
            THEN sqlia IS severe;

RULE 10 : IF attack_type_severity IS high AND stmnt_risk_level  IS ddl
            THEN sqlia IS severe;
```

Fig. 5.4: Fuzzy inference rule set

5.2 IMPLEMENTATION

Consider the different kinds of input values in a SQL query as shown in fig 5.5. First one is a valid query despite containing quotation mark. The second form of input leads to breach in confidentiality of the system while the last one results in the loss of sensitive data.

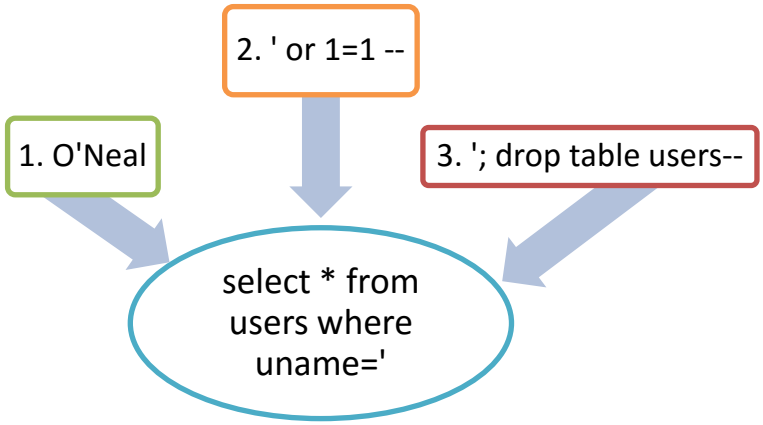

Fig 5.5: Various sorts of input to a query

Initially, the negative taint values of all inputs are calculated followed by determining the type of query, in this case, a DML. Lastly, these values are fuzzified to indicate SQLIA. Our system shows appropriate indications for threat levels produced by the different inputs as shown in Fig 5.6, 5.7, 5.8.

Client Side View:

Login Success!

Account Details

Welcome O'Neal

First Name Shaun
Last Name O'Neal
Gender M
Email shaun@gmail.com

Actions Privileges

Server Side View:

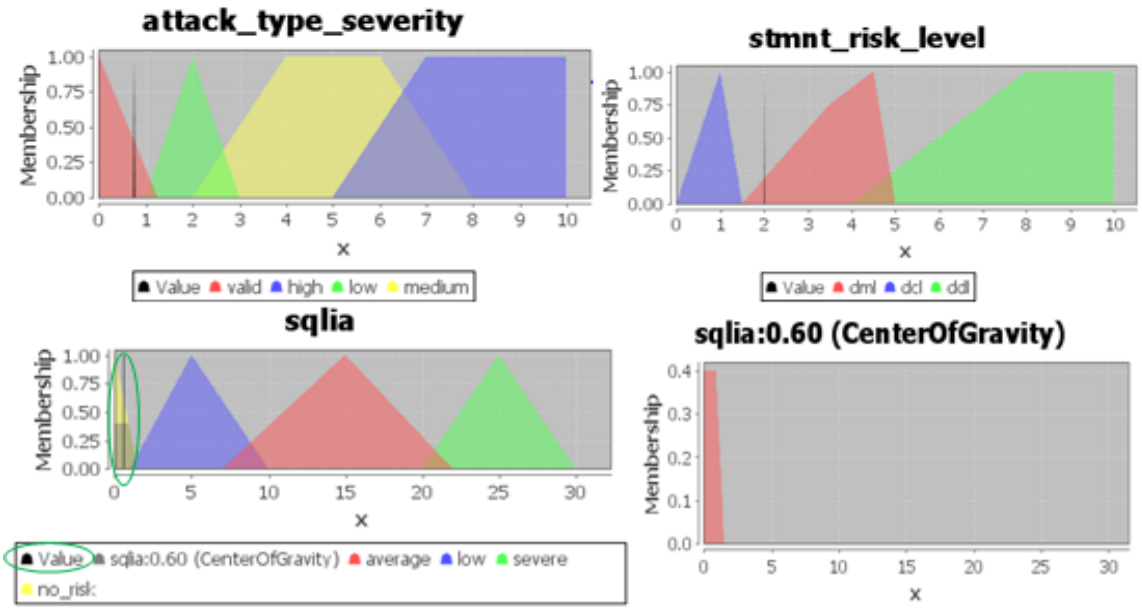

Fig. 5.6: Input > O'Neal

Client Side View:

Invalid user ' or 1=1 --

Disclaimer : SQLIA detected. Can't Proceed further. Repeated attempt might lead to IP blocking.

Server Side View:

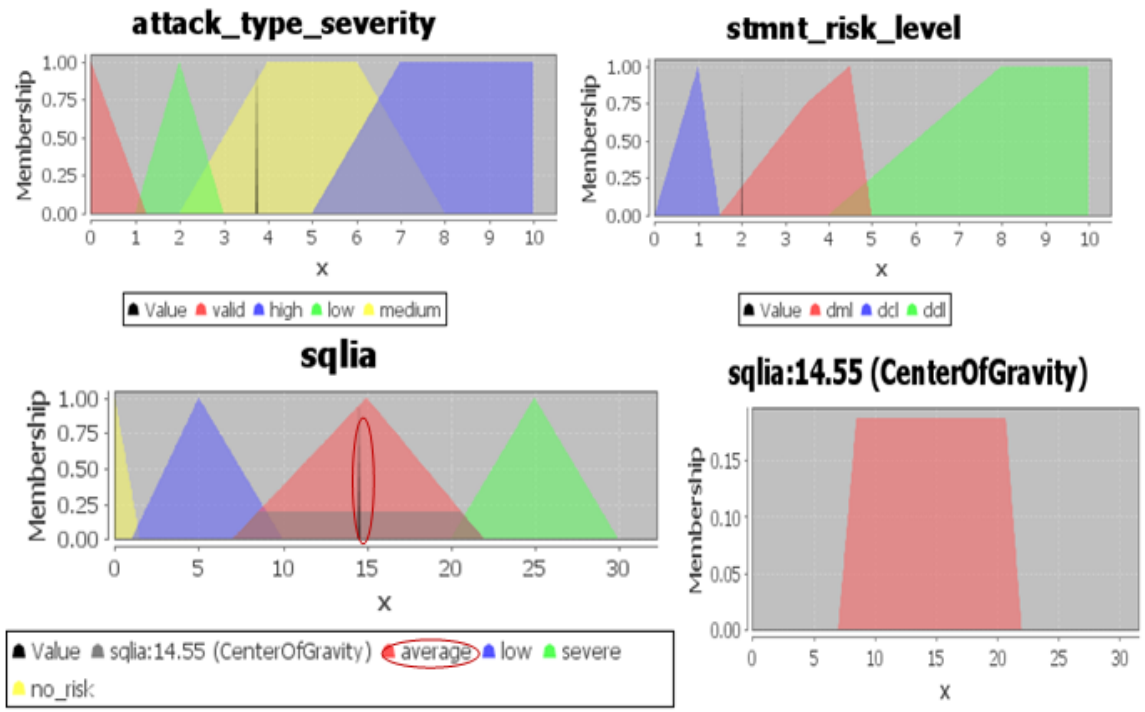

Fig. 5.7: Input > ‘ or 1=1 --

Client Side View:

Invalid user '; drop table users --

Disclaimer : SQLIA detected. Can't Proceed further. Repeated attempt might lead to IP blocking.

Server Side View:

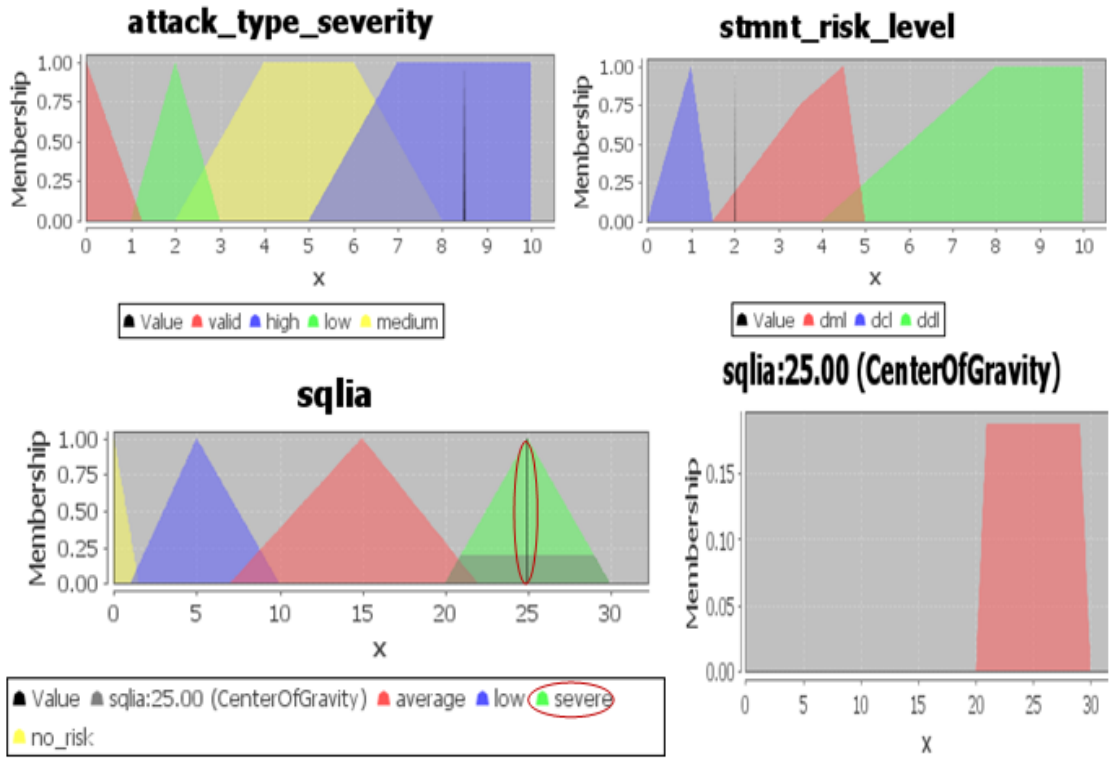

Fig. 5.8: Input > '; drop table users--

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

This thesis work has focused on thwarting the SQL injection attacks with the help of a well known mathematical model known as fuzzy logic. Initially, the various types of existing techniques used to detect or prevent SQL injection where studied to find places for improvement and the attacks were simulated on a dummy website to get the insights about the hackers strategies to perform SQL injection. Finally after many experimentations and brainstorming for ideas, a model was proposed to detect the impact of a full-fledged query on the data source and works on successfully removing false positives. This approach uses the commonly known COG method while implementation of other methods can be explored in the near future. Security of the application can be enhanced by including more encoding patterns. This method can further be extended to solve the issue of No SQL injection.

REFERENCES

[1] "2015 Cyber Attacks Statistics – HACKMAGEDDON by Paolo Passeri." [Online]. Available: http://www.hackmageddon.com/2016/01/11/2015-cyber-attacks-statistics/. [Accessed: 28-Mar-2016].

[2] the free encyclopedia From Wikipedia, "Database." [Online]. Available: https://en.wikipedia.org/wiki/Database. [Accessed: 10-Apr-2016].

[3] the free encyclopedia From Wikipedia, "SQL." [Online]. Available: https://en.wikipedia.org/wiki/SQL. [Accessed: 10-Apr-2016].

[4] "What Are Web Applications?" [Online]. Available: http://www.acunetix.com/websitesecurity/web-applications/. [Accessed: 10-Apr-2016].

[5] "What is Web application (Web app)? - Definition from WhatIs.com." [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app. [Accessed: 10-Apr-2016].

[6] "What is a Web Application?" [Online]. Available: http://webtrends.about.com/od/webapplications/a/web_application.htm. [Accessed: 10-Apr-2016].

[7] T. J. Ross, *Fuzzy Logic with Engineering Applications*. 2004.

[8] S. RAJASEKARAN and G. A. V PAI, *NEURAL NETWORKS, FUZZY LOGIC AND GENETIC ALGORITHM: SYNTHESIS AND APPLICATIONS (WITH CD)*. PHI Learning, 2003.

[9] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao, "A static analysis framework for detecting SQL injection vulnerabilities," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 87–94, 2007.

[10] A. Orpani, "Joomla SQL Injection Vulnerability Exploit Results in Full Administrative Access." [Online]. Available: https://www.trustwave.com/Resources/SpiderLabs-Blog/Joomla-SQL-Injection-Vulnerability-Exploit-Results-in-Full-Administrative-Access/?page=1&year=0&month=0. [Accessed: 10-Apr-2016].

[11] T. Seals, “SQL Injection Possible Vector for TalkTalk Breach - Infosecurity Magazine.” [Online]. Available: http://www.infosecurity-magazine.com/news/sql-injection-possible-vector-for/. [Accessed: 26-May-2016].

[12] “One of the Largest Hacks Yet Exposes Data on Hundreds of Thousands of Kids | Motherboard.” [Online]. Available: http://motherboard.vice.com/read/one-of-the-largest-hacks-yet-exposes-data-on-hundreds-of-thousands-of-kids. [Accessed: 10-Apr-2016].

[13] “Top 10 2013-Top 10 - OWASP.” [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10. [Accessed: 02-Mar-2016].

[14] “CWE - 2011 CWE/SANS Top 25 Most Dangerous Software Errors.” [Online]. Available: http://cwe.mitre.org/top25/. [Accessed: 02-Mar-2016].

[15] “2015 Web Application Attack Report (WAAR),” 2015.

[16] Verizon, “2015 Data Breach Investigations Report,” *Inf. Secur.*, pp. 1–70, 2015.

[17] Trustwave, “2015 Trustwave Global Security Report,” p. 90, 2015.

[18] “Vulnerability Case Study: Cookie Tampering.” [Online]. Available: http://www.infosectoday.com/Articles/Cookie_Tampering.htm. [Accessed: 10-Apr-2016].

[19] W. G. J. Halfond, J. Viegas, and A. Orso, “A Classification of SQL Injection Attacks and Countermeasures,” 2006.

[20] Y. Huang, S. Huang, T. Lin, and C. Tsai, “Web application security assessment by fault injection and behavior monitoring,” *Proc. 12th …*, pp. 148–159, 2003.

[21] S. W. Boyd and A. D. Keromytis, “SQLrand: Preventing SQL injection attacks,” *Appl. Cryptogr. Netw. Secur.*, pp. 292–302, 2004.

[22] Y.-W. 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,” *Proc. 13th Int. Conf. World Wide Web*, pp. 40–52, 2004.

[23] W. G. J. Halfond and A. Orso, “Combining static analysis and runtime monitoring to counter SQL-injection attacks,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[24] Y. Shin, L. Williams, and T. Xie, “SQLUnitGen : Test Case Generation for SQL

Injection Detection 1 Introduction 2 Background."

[25] K. Kemalis and T. Tzouramanis, "SQL-IDS: A Specification-based Approach for SQL-injection Detection," *Proc. 2008 ACM Symp. Appl. Comput.*, pp. 2153–2158, 2008.

[26] W. G. J. Halfond, A. Orso, and P. Manolios, "{WASP:} Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation," *{Ieee} {Tse}*, vol. 34, no. 1, pp. 65–81, 2008.

[27] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, pp. 14:1–14:39, 2010.

[28] S. Roy, A. K. Singh, and A. S. Sairam, "Detecting and Defeating SQL Injection Attacks," vol. 1, no. 1, 2011.

[29] A. S. Gadgikar, "Preventing SQL injection attacks using negative tainting approach," *IEEE Int. Conf. Comput. Intell. Comput. Res.*, pp. 1–5, 2013.

[30] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, "Automated testing for SQL injection vulnerabilities: an input mutation approach," *Proc. 2014 Int. Symp. Softw. Test. Anal. - ISSTA 2014*, pp. 259–269, 2014.

[31] A. Sadeghian, M. Zamani, and A. Abd. Manaf, "SQL injection vulnerability general patch using header sanitization," *I4CT 2014 - 1st Int. Conf. Comput. Commun. Control Technol. Proc.*, no. I4ct, pp. 239–243, 2014.

[32] B. Hanmanthu, B. R. Ram, and P. Niranjan, "SQL Injection Attack prevention based on decision tree classification," *Intell. Syst. Control (ISCO), 2015 IEEE 9th Int. Conf.*, pp. 1–5, 2015.

[33] P. Cingolani and J. Alcalá-Fdez, "JFuzzyLogic: A robust and flexible Fuzzy-Logic inference system language implementation," *IEEE Int. Conf. Fuzzy Syst.*, 2012.

[34] P. Cingolani and J. Alcala-Fdez, "jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming," *Int. J. Comput. Intell. Syst.*, vol. 6, no. sup1, pp. 61–75, 2013.

[35] "IEC 61131-7 First edition Reference number CEI/IEC 61131-7:2000", *Internationale International Standard*, pp. 17–57, 2000.

[36] "Fuzzy control," *Int. Electrotech. Comm. Tech. Comm. No. 65 Ind. Process Meas.*

Control SUB-COMMITTEE 65 B DEVICES, no. 65, pp. 1–53, 1997.

ANNEXTURE I

LIST OF PUBLICATIONS

Khanna Surya, Verma A.K, "Classification of SQL Injection Attacks Using Fuzzy Tainting**"** *4th International Conference on* Advanced Computing, Networking, and Informatics (Springer) [ICACNI - 2016], National Institute of Technology, Rourkela, India **[Accepted and Registered]**

ANNEXTURE II

VIDEO PRESENTATION

Live video link: https://youtu.be/Ir1Pgu5aOF0

ANNEXTURE III

PLAGIARISM CERTIFICATE

801433028_Surya

ORIGINALITY REPORT

4%	2%	3%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.cc.gatech.edu Internet Source	1%
2	"Search-Based SQL Injection Attacks Testing Using Genetic Programming", Lecture Notes in Computer Science, 2016. Publication	<1%
3	Shyi-Ming Chen. "NEW METHODOLOGY TO FUZZY REASONING FOR RULE-BASED EXPERT SYSTEMS", Cybernetics and Systems, 3/1/1995 Publication	<1%
4	Li, Xiaowei, and Yuan Xue. "A survey on server-side approaches to securing web applications", ACM Computing Surveys, 2014. Publication	<1%
5	www.slideshare.net Internet Source	<1%
6	Submitted to Madan Mohan Malaviya University of Technology Student Paper	<1%