



LOVELY
PROFESSIONAL
UNIVERSITY

**Prevention and Detection of SQL Injection Attacks using an Advanced Version
of Aho-Corasick Algorithm**

A Dissertation Report

Submitted By

Nitika

(11001664)

To

Department of Computer and Engineering

In fulfillment of the Requirement for the

Award of the Degree of

Master of Technology in Computer Science

Under the guidance of

Mr. Roshan Srivastava

(May, 2015)

PAC FORM

ABSTRACT

In, today's internet world we have dependent on online website applications like email, e-commerce, social networking sites, online banking, blogs, forms, online banking, blogs, forums and many more. As we know that web application uses databases as a backend. As the demand of the online web application increases threats to websites are also increasing. OWASP (Open Application Security Project) listed top ten attacks every year and it listed SQL injection as a top most security attack. Security of our database is more important because our database consist of many important data so the loss of the important data will result into big loss. Many researchers work on the prevention of SQL injection attacks and gives different methods of detection and prevention. Here In chapter 1 we have given introduction of SQL injection attacks. In chapter 2 we have discussed about the different techniques used to prevent and detect SQL injection attacks. Many researchers have been given there methodologies and discuss about what are their limitations advantages of that particular methods which they have used. In chapter 3 we have discussed about scope of our study. In chapter 4 we have discussed about objective of our study. In chapter 5 we have discussed our Research methodology. In chapter 6 we have discussed about our result. In chapter 7 we have described summary and conclusion.

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of our research work depends largely on the encouragement and guidelines of many others. We take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I would like to show my greatest appreciation to my research work mentor, **Mr. Roshan Srivastava** I can't say thank you enough for the tremendous support and help. I felt motivated and encouraged every time I attended his meeting. Without his encouragement and guidance this project work would not have materialized.

I'm highly grateful to **Mr. Dalwinder Singh**, Head of Department, for his thorough guidance right from day 1 to the end of research work. He actually laid the ground for conceptual understanding of research work.

Nitika

DECLARATION

I hereby declare that the dissertation2 entitled, “**Prevention and Detection of SQL Injection attacks using an Advanced Version of Aho-Corasick Algorithm**” submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date:-

Investigator:-

Registration No:-11001664

CERTIFICATE

This is to certify that Nitika has completed M.Tech dissertation2 proposal titled “**Prevention and Detection of SQL Injection attacks using an Advanced Version of Aho-Corasick Algorithm**” under my guidance and supervision. To the best of my knowledge, the present work is the result of her original investigation and study. No part of the dissertation proposal has ever been submitted for any other degree or diploma.

This dissertation2 proposal is fit for the submission and the fulfilment of the condition for the award of M.Tech computer science and Engineering.

Date:

Name:

UID:

TABLE OF CONTENTS

S. No	Topic	Page No.
1	Introduction	1
2	Categories of SQL injection attacks	7
3	Types of SQL injection attacks	8
4	Prevention of SQL injection attacks	9
5	Post-Generated Approach	9
6	Pre-Generated Approach	10
7	Detection of SQL injection attacks	11
8	Review of Literature	13
9	Scope of Study	17
10	Objective of Study	21
11	Formulation of Hypothesis	22
12	Research Design	22
13	Pseudo code of Anomaly pattern matching algo	23
14	Calculation of Pattern score	25
15	PFAC algorithm	25
16	Results and Discussion	27
17	Future scope and conclusion	37
18	References	38

LIST OF FIGURES

S. No	Figure	Page No.
1	User form with valid username and password	2
2	User form with invalid username and password	2
3	SQL injection attacks in past 12 months	4
4	Company facing SQL injection attacks today	4
5	Increasing state of SQL injection attacks	5
6	Web application Firewall	6
7	Shows company familiarity with SQL injection attacks	7
8	Original query	14
9	Injected query	14
10	Goto Function	19
11	Failure Function	19
12	Output Function	19
13	Failure Function of Advanced version of A-C	20
14	Thread Based Technology	23
15	Proposed Architecture	24
16	Shows anomaly pattern for user input queries	27
17	Shows stored anomaly patterns for user anomaly queries	28
18	Shows user input queries	29
19	Shows anomaly patterns related to user input queries	30
20	Shows matched anomaly patterns	31
21	Shows user input anomaly patterns	32
22	Shows output of anomaly patterns	33
23	Shows successfully added patterns	34

LIST OF TABLES

S. No	Topic	Page No.
1	Calculation of Pattern Score for vulnerable queries	35
2	Comparison of our techniques with various schemes	36

CHAPTER 1 INTRODUCTION

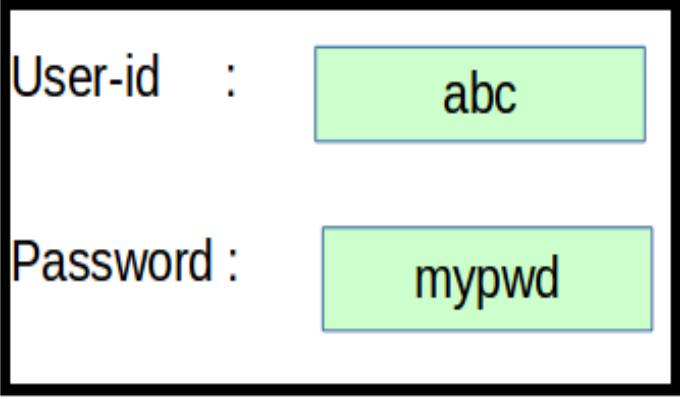
SQL is a language for communicating and manipulating data held in relational database management systems (RDBMS). SQL consists of data manipulation commands and data definition commands. SQL was first commercial language for E.F Codd's relational model. SQL injection is a code injection technique, used to attack data driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL Injection Attacks are most effective method for stealing data from back-end database, by the help of these attacks hacker can get access to the database and steal sensitive information. Generally these attacks are generated from web input so these attacks are called input validation attacks. Now a day's most web applications are being hacked using SQL Injection attacks method. It comes under top ten security threat in web applications. A successful SQL injection can leak our important data and unauthorized user can get our database access and he or she can modify our database, here modify means he or she can insert, update, and delete our important data. A successful SQL injection can lead to following:

- Malicious user can change the database data by using data manipulation language (DML) commands.
- Malicious user can perform all the operation which is performed by administrator.
- Recover the content of a file which is present on the DBMS file system
- Extract useful information from the file system of the database.

Occurrences of SQL injection Attacks:

- Confidentiality: It shows how much the data is encrypted or secure inside the database.
- Authentication: It defines how much an untrusted and unauthorized person is restricted.
- Authorization: It checks the valid users of the systems and their privileged.
- Integrity: The alteration on dataset is restricted to privileged persons only.

Example of SQL injection



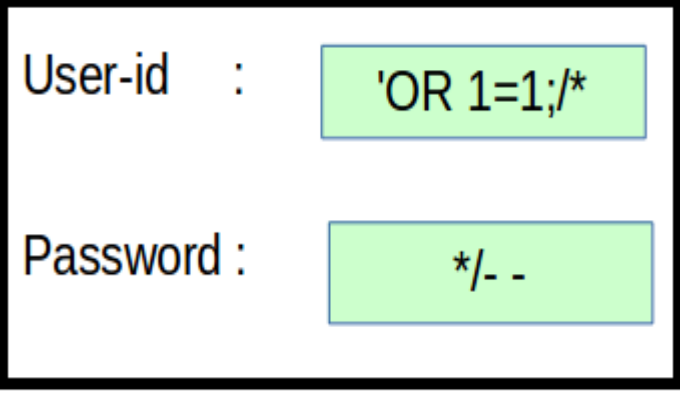
User-id : abc

Password : mypwd

Figure 1: Userform with valid username and password

SQL query for Figure1: `Select * from my_table where userid='abc' AND password='mypwd';`

This query is legitimate query



User-id : 'OR 1=1; /*

Password : */- -

Figure 2: Userform with invalid username and password

Vulnerable query

SQL query for figure2: `Select * from my_table where userid =' ' OR 1=1; /*/ and password='*/- -'`

The most salient findings are shown below:-

- The SQL injection attacks are taken seriously because 65% of organizations are experienced a SQL injection attacks.
- Almost 49% of respondents say the SQL injection attacks experiencing their company or organization is Very significant. On average, respondents believe 42 percent of all data steals are due, at Least in part, to SQL injections.
- Organizations are not familiar with the techniques used by cyber criminals. 46% of respondents are familiar with the term Web Application Firewalls (WAF) Bypass. Only 39% of respondents are familiar with the techniques cyber Criminal use to get around WAF perimeter security devices.
- BYOD makes understanding the root causes of an SQL injection attack more difficult. 56% of respondent said that root cause of SQL injection attacks are very difficult to know.
- Expertise and the right technologies are not able to prevent SQL injection attacks. While respondent concludes that SQL injection attacks are very serious security threat. Only 35% of organization agrees that they have right tools and technologies to detect a SQL injection attacks.
- Prevention of SQL injection attacks are lacking as 52% organizations are not validating third party software.

Key findings:-

In this section we provide an analysis of key findings .we will talk about the followings:

- Scope of the SQL injection threat
- Why Organization remain vulnerable to SQL injection attacks.

Scope of the SQL injection threat

As shown in Figure 3, 65% of organizations are experiencing SQL injection attacks that are not even controlled it by firewalls and other defensive techniques. 40% respondent say that it takes very long time to detect SQL injection attack or not able to detect SQL injection attacks. Average time is around 68 days to detect.

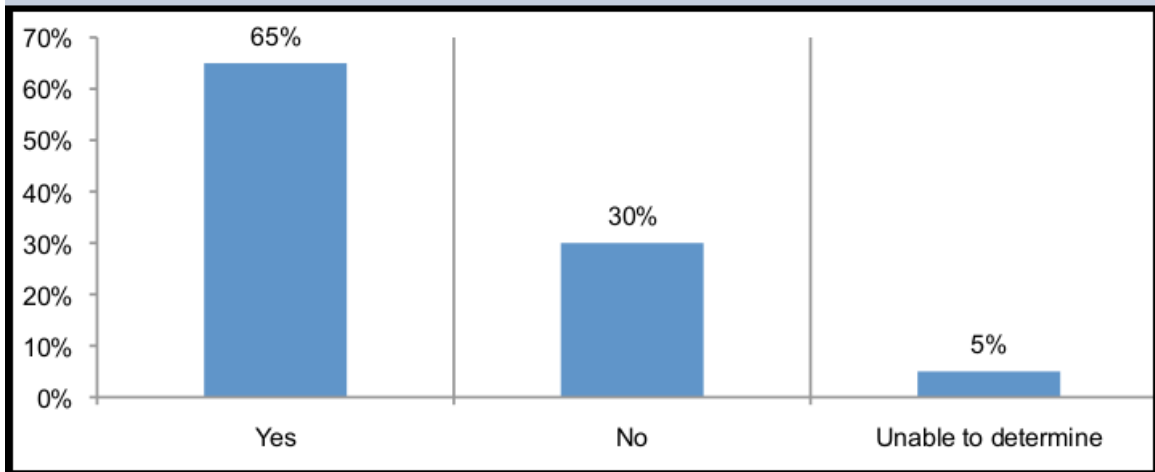


Figure 3: SQL injection attacks in past 12 months

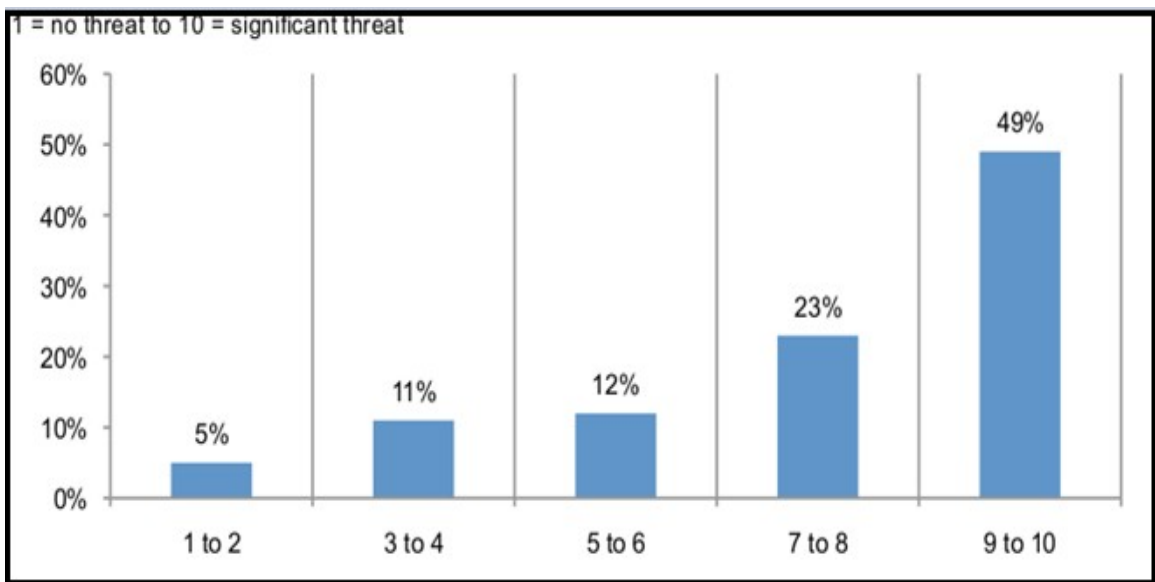


Figure 4: Company facing SQL injection attacks today

The SQL injection attacks are taken seriously by 50% of organizations as shown in figure3. Only 5% of organizations say that SQL injection is not a serious threat to their organizations. SQL injection attacks are increasing day by day Figure3 shows the increasing state of SQL injection attacks. 45% respondent says that the states of the SQL injection attacks are at same level. 38% respondent says that the SQL injection attacks are

Increasing very rapidly. 13% respondent says that SQL injections attacks are decreasing.

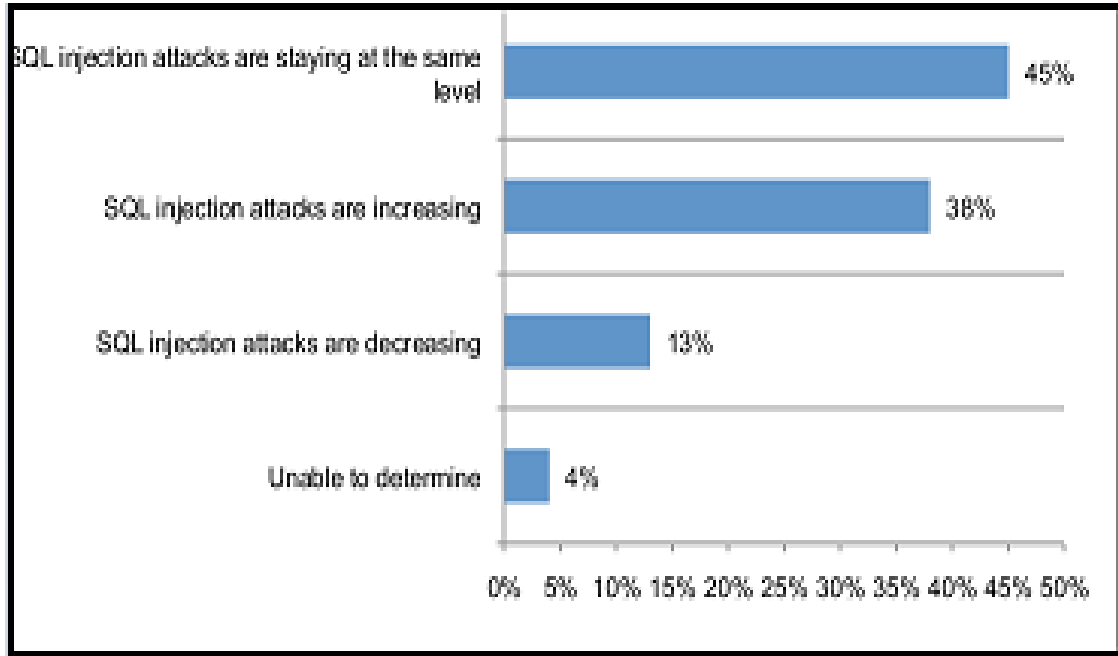


Figure 5: Increasing state of SQL injection attack

Why Organization remains vulnerable to SQL injection attacks: - Organizations are not familiar with new techniques used by attackers. Only 46% respondents are familiar with WEB APPLICATION FIREWALLS (Figure 6) bypass. As shown in Figure 4, only 39% of organizations are familiar with the techniques cybercriminal use to get around WAF perimeter security devices.

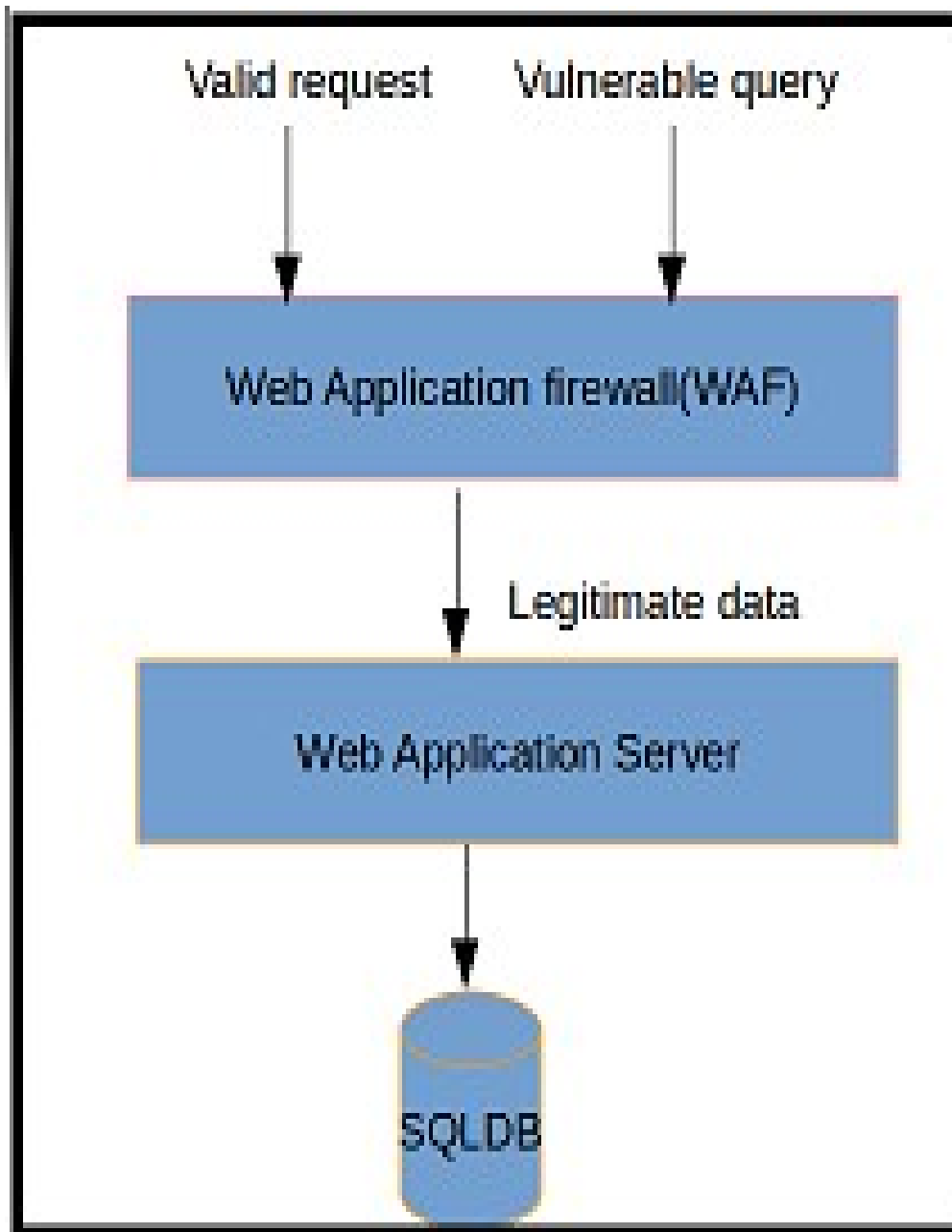


Figure 6: Web Application Firewall

Web Application Firewall is used to prevent SQL injection attacks, XSS attacks, Brute Force attacks, Distributed Dos. It protects from various security threats listed by OWASP. It can detect cookie, session and parameter tampering. WAF is like a normal firewall but difference is that it is specially designed to specific HTTP server such as Apache. It is not able to mitigate all types of attacks like hijacking of sessions.

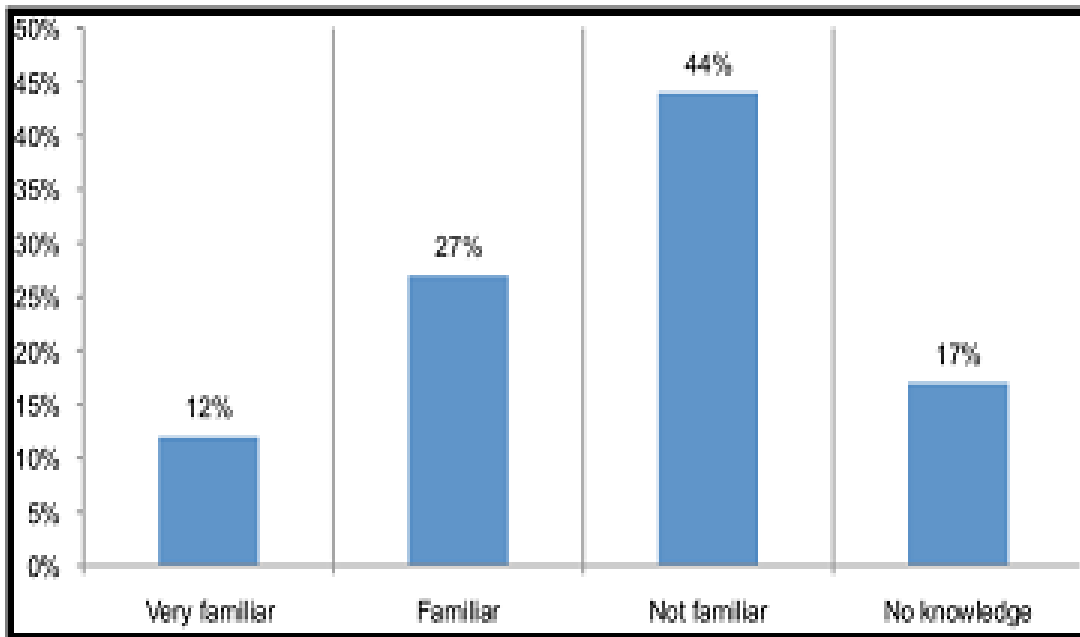


Figure 7: Shows company familiarity with SQL injection attacks

Figure7: shows that 19% financial services, 12% public sector, 10% of technology and software industry web applications are vulnerable to SQL injection attacks. 9% of Health and pharmaceuticals and 8% of web applications are vulnerable to SQL injection attacks.

1.2 Categories of SQL Injection Attacks

SQL Manipulation : It is the process of modifying the SQL statements by using various operations such as UNION another way of implementing SQL Injection using SQL Manipulation method is by changing the where clause of the SQL statement to get different results.

Code Injection: Code injection is the process of inserting new SQL statement. One of the code injection attacks is to append a SQL Server EXECUTE command to the vulnerable SQL statement. This type of attack is only possible when multiple SQL statements per database request are supported.

Function Call Injection: It is the process of inserting various database function calls into a vulnerable SQL statement. These function calls can make an operating system call or

manipulate data in the database.

Buffer Overflow: Buffer overflow is caused by using function call injection. For most of the commercial and open source databases, patches are available. This type of attack is possible when the server is un-patched.

1.3 Types of SQL Injection Attacks

There are many types of attacks which are used together by attackers. The attacks are: -

Tautologies

In this attack the code is transfer to conditional statements. This result true and these attacks are used in conditional statement such as WHERE.

Illegal/incorrect queries

In this attack the attackers uses the back-end information of database. For that information of the attackers uses the error page to get the parameter. That can be changed or edit. By these the attackers get the information about tables.

Union Query

In union query the attackers use the two queries to get the information. The attacker find the vulnerable argument to change the particular result set. This query help the attacker to get information about specified table and the attacker make changes to get information.

Piggy Backed Queries

In this type of attack the attacker add many queries to the original. In this attack all the queries are executed. By using this attack the attacker get the information about database. In this attack the attacker used stored procedure with the queries.

Stored Procedures

In this attack the attacker used different stored procedure for execution. If the attacker successful in getting back-end database information, he can use stored procedure for attacking purposes.

Inference

In this attack the attacker uses true or false question to get data information. In these

attack attackers uses commands to get information about a site. The attacking strategies of attacker are based on observation.

There are two attack techniques based on inference.

- **Blind injection:** In this the attacker uses true or false questions and after this the pages are changes to functionality.
- **Timing Attacks:** In this the attack the attacker observes the website very carefully. Attacker observes the timings and by using some queries he can find data.

Alternate Encodings

In this attack the attacker modified the code to avoid the techniques which detects the attacks. In this attack the attacker alters the text which is injected in the site. The attacker also used quotes and operators also. Due to prevention techniques the text is detected which contains SQL injection attacks to avoid that that modifications in code are done.

1.4 Detection of SQL Injection Attacks

There are many approaches for the detection of SQL injection attacks and it can be categorized into Pre-generated approach and Post-generated approach and Post-generated approach. Before sending the query to the database server analysis of the query is to be done. If the query comes out to be vulnerable query then it must be blocked. Static approaches are done on training phase. Web application need to secure from SQL injection attack (SQLIA). Pre-generated approach is very effective in nature to check the input validation of data. Following are some Post-generated and Pre-generated.

1.4.1 Post Generated Approach

There are three popular detection techniques of SQLIA using post-generated approach.

Positive Tainting and Syntax Aware Evaluation:

In this approach input strings are passed to the system for the detection of SQL injection attacks. It filters the input on the bases of hard coded strings, strings that are implicitly created by Java. At runtime this filter process occurs. If any types of untrusted strings are found then it is blocked and it is restricted to pass to the database server. The strings are categorized as:

- (i) Hard coded strings
- (ii) Strings implicitly created by Java
- (iii) Strings originated from external sources.

In case of syntax aware evaluation it is done at the database interaction point. Syntax are defined according to the policies of the defined function for the detection of SQL injection attacks. The main work of the defined function is to check the pattern that whether the inputted data is valid to send to the database server or not. Developers can initialize the trusted strings but limitation is that storage of strings can cause second order attack.

Context Sensitive String Evaluation (CSSE):

The basic idea behind this approach is to find the root cause of the SQL injection attacks, which means that from where the untrusted data are coming whether from the user side or from the developer side like improper input validation. As we know that vulnerabilities can also occur from the developer side. In this approach syntax analysis is done. Basically it first distinguishes between the numerical constants and strings constants. This operation is performed before sending the query to the database server. Following issues are there in this approach.

- i) Unsafe character initialization is dependent on the developers.
- ii) Removal of unsafe characters restricts the application functionality.

Parse tree evaluation based on grammar:

In this approach those queries are blocked to send to the database server those do not follow syntactic structure. At runtime a parse tree is generated on the basis of the pre-defined grammar. Special characters are specially matched. If the queries do not follow the syntactic structure. If the query parser successfully, which means it is legitimate query. Otherwise it is vulnerable query.

1.4.2 Pre-Generated Approach

Pixy: A Static Analysis Tool:

Pixy is a tool for checking the vulnerabilities of web application. Data analysis is done to

check that whether it is entered by malicious user or not. Assumption is done in this approach that SQL injection attack is done due to some parameters. Firstly author identifies all vulnerable parameter. After identifying parameter scheme for removing the vulnerable parameter is needed. A literal analysis is also done in this tool to check about the literals values and constant values. Limitation of this tool is that it is an open source tool. An attacker may try to get new technique to bypass.

Program Query Language (PQL):

A PQL is specially designed to deal with attack related queries. It uses static technique to check the solution of such attacks. It finds all matching related to the context sensitive as well as flow insensitive analysis. The results from the static analysis are further send to dynamic analysis. It uses pre-defined grammar. Limitation of this tool is that it is based on output and it is mostly depend on the developer.

1.5 Prevention of SQL injection attacks

There are various techniques and frameworks are developed by various researchers to prevent SQL injection attacks.

Learning Database Design

Reducing Permissions: Reducing or Limiting permission is very good way to prevent SQL Injection attacks. Limited permission means only necessary permission is to be given to the user account as we know that user account is used to access database and restriction to the application level layer is always good choice. If the security is not good at this level than the user may use some bypassed query and get access to the database. Several accounts are used by many people to connect to database. It is not good choice because different privileged are there for different accounts.

Defensive coding practices

Programmers must write defensive coding which means proper input validation is there like web application contains forms which consist of various fields like name, phone number, addresses, and email. On the fields input validation can be provide by writing validation checking code in the program. And some inbuilt functions can be used. Error

page which is displayed, it creates problem which tells about the database schema.

- All input sources must be verified
- All input sources must be verified that whether the data is coming from the trusted source or not and if it is found that untrusted source is sending data it must be blocked. Basically defensive coding practices are best way to prevent SQL injection attacks. But it is not possible that it will be applicable to automated techniques. It is better to use stored procedure and prepared statements. But it is not necessary that by using this SQL injection attacks can be successfully prevented.

Pattern Matching

Pattern Matching is very effective way to prevent SQL injection attacks in which vulnerable patterns are already stored and thus comparison of the incoming query with the already stored query is done.

CHAPTER 2 REVIEW OF LITERATURE

Prithvi Bisht et al [8] in this paper, author proposes a tool in which whenever the programmer planned a program for checking SQL injection attacks blocks in program designed in such a way that each block number identifies some kind of attack. Basically this scheme is called control path and which is used in Oracle. Since it is not possible to construct Oracle. Author introduces a procedure named comparison of parse tree SQL. Execute command is used to call the procedures for checking that inserted query is correct query or injected query. This tool is specially written for Java applications. In the proposed method parse tree for both queries are compared.

Srinivas et al [9] uses the concept of randomization in which the input query is converted to cipher text using Random4 algorithm. So the malicious user is not able to identify detail like table name, column name. Author considers the valid input as numbers, lowercase, uppercase characters and 10 special characters. A look up table is introduced for each character, numbers, Special characters four random values are assigned.

Stephen W. Boyd et al [10] discussed about the concept of randomization. SQL keywords are manipulated by appending integer to them. The integer values are not easily guessed by the malicious users. Therefore any malicious user attempting an SQL injection attack can be detected. Concept of proxy is there and proxy is placed between web server and database server. Proxy behaves as a virtual database. Unique random key is used by proxy to decode the SQL commands which is known as derandomization.

Shelly Rohilla at al [11] in this paper, author proposed security for both back-end and for front-end. Authors used a concept of log entry. Whenever illegal user enters a log entry will be generated in which date and time are included. And also IP address of that user is also saved. In back-end attack is prevented by using stored procedure.

AS.Gadgikar et al [1] in this paper, author proposes a new approach known as negative tainting which means all known SQL injection attacks are checked for every query inputted by the user. Author proposed a method in which already known attacks are stored

in the form of primary linked list and divided in the form of tokens. Procedure is being used to convert the tokens into integer numbers and all are stored in the secondary linked list.

MA.Amutha Prabakar et al [3] in this paper authors discussed about prevention and detection of SQL injection attacks using A-C Algorithm. Author used pattern matching algorithm. Aho–Corasick Multiple keyword matching algorithm A-C is a multiple string matching algorithm developed by Alfred. The complexity of the algorithm is linear. This algorithm is based on DFA (Deterministic finite automata) basically the ac function consists of three functions goto function g, failure function f, output function which is simply used for the transition from one state to another state.

Debabrata Kar et al [2] proposed a scheme of query transformation Scheme that transforms the query into structural form rather than parameterized form. Author assumes that any query which is coming is equally vulnerable to injection attack. Author converts the queries into same structural form. Therefore we need to store only one structure form instead of two skeleton queries. The structural form of the injected query is different from stored structure. So we can easily determine the injected query. Advantage of this technique is that it can reduce number of different queries into same structure form, minimizing the size of the search space for run –time matching.

Query Transformation scheme

Query is transformed into its structural form rather than its parameterized form and query can be transformed by using the preg_replace function () which is easily available in php Hashing is performed on the transformed query. In order to store number of transformed query efficient searching is also needed during runtime for this suitable hash function is used. In this paper md5 hashing function is used.

Jeom Kim et al [5] in this paper, author proposed detection of SQL injection using combination of static and dynamic analysis. Detection method is performed by removing the values of the attribute of SQL queries. For this author proposed an algorithm in which delete function is used to remove the values of the attributes of SQL query. Finally, query with deleted attributes values and fixed query is compared by using XOR operation. If the result of the both query is 0 means it is non-injected query otherwise query it is

vulnerable query.

Michelle Ruse et al [6] in this paper, author main idea is to create a model. By using that model comparison of queries are done. A tool is made for detection and prevention of SQL injection Attacks and the tool is written in C programming language.

Ntagwabira Lambert et al [7] discussed about prevention and detection of SQL injection attacks using Query tokenization. Author discussed a method which detects a before a space, single quote, space, before double dashes, double quotes makes a token. Basically author grouped all the tokens together and put them into the array that tokens play as an index. This process is known as tokenization. Tokenization is done for both the injected query as well as for original query. If the length of the both query differs than an injection is there. Otherwise no injection is there. Examples are:-Original Query

Select * from table where attribute = 'userinput'



Figure 8: Original query

Injected Query

Select * from table where attribute = 'user input' or '1'='1'

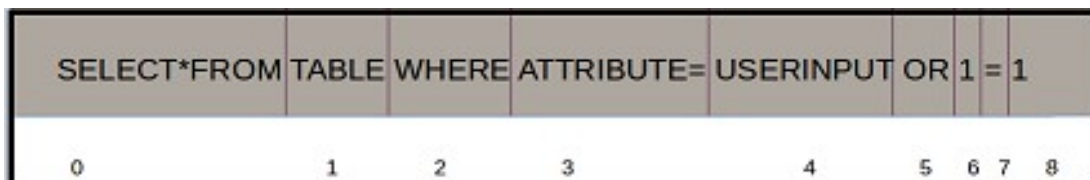


Figure 9: Injected query

If we compare the two arrays described in the Figure8 and figure9, we observe that the length of the injected query or vulnerable query is greater than the original query. In this paper author main idea is token formation. Author methodology is that they made Query

parser method. The Query parser method is executed for both the two queries Vulnerable query and original query each of which is considered as a text string. The Tokenization is done for every string, detecting space, single quote, double dashes. Secondly Query parser read each input character by character and whenever it reaches spaces, double dashes, and single quote. It creates a token for that and when the tokenization is finished for both queries query which is injected query and query which is original query are arranged in the form of array and lengths of both of the queries are calculated. If there is difference in the length of the original query and the injected query it means the injection attack is there.

G.Aghila et al [4] in this paper, author checks that whether the coming data is received from the trusted source or not and not valid resources can be easily eliminated. Author used Hirschberg algorithm which uses a method divide and conquer for the purpose of token comparison. Author maintains a table in which the values are predefined by the programmer and the values are compared to incoming tokens. This technique is applied on banking applications.

William G.J Halfond et al [12] discussed about tool named “AMNESIA” This tool firstly identify hotspot, and then build SQL Query Models. JSA which converts NDFA to its corresponding DFA and for each hotspot call to monitor is sent before calling database. The monitor is invoked on the basis of following parameters.

- 1) The string that contains the actual query which is to be submitted and unique identifier for the hotspot. Because of the unique identifier, the runtime monitor is able to identify hotspot with particular SQL query. This tool is tested for seven web applications.

CHAPTER 3 SCOPE OF STUDY

As we know that pattern matching plays very important role in computer science. Pattern matching is used to identify anomaly or odd one packet from all the packets. With the increase of Internet most of the off-line works are converted into on-line works. There are many algorithms used for pattern matching which are following.

Karp-Rabin algorithm: In this algorithm we need a hash function having different hash values for substrings. ASCII codes are used in hash function. The complexity of the Rabin Karp algorithm is $O(nm)$ where n is the length of the text and m is the length of patterns. It is bit faster than Brute force string matching. In practice its complexity is $O(n+m)$.

Brute force algorithm: Brute force string matching algorithm uses very simple concept. Two main sets are required one is text and second is pattern. First character of the pattern is matched with the first character of text. If it is matched then fine otherwise move forward. The complexity of the brute force search is $O(nm)$. Here n is the length of the text and m is the length of the pattern. It is quite slow whenever large text is there.

Knuth-Morris-Pratt algorithm: As we know that brute force algorithm and Rabin Karp algorithm both are not much effective. Brute force algorithm is very slow to remove slow nature of Brute force algorithm hash function is introduced in Rabin Karp algorithm. In Morris Pratt algorithm pre-process of pattern is done. Comparison is done if any mismatch occurs then from table it checks the position of character. If a letter is repeated in the pattern list and if mismatch occurs after that letter, we will compare it from the position from the table. This algorithm takes some time and space. But it pre-processes the pattern in $O(m)$ where m is the length of pattern. And it takes searching time $O(n+m)$ where n is length of text and m is the length of pattern. Here we will do pre-process only once and search the patterns.

Boyer-Moore algorithm: Boyer Moore algorithm is faster as compared to previous algorithm it uses the concept of good suffix and bad character shifts to improve the performance of the algorithm. Its worst case complexity is $O(n+m)$ where n is the length of the text and m is the length of patterns.

Commentz Walter algorithm: This algorithm is a combination of A-C algorithm and Boyer Moore algorithm. By combining average case is improved. Worst case is $O(mn)$ where m is the length of text and n is the length of patterns.

Aho-Corasick algorithm: A-C algorithm is a classic algorithm for exact string matching. Its running time is $O(n+m+z)$ where n is the length of the text, m is the length of patterns, and z is the no of occurrences of patterns. A-C algorithm uses two phases first phase is construction of finite state machine and second phase is searching phase. Searching of patterns in the text. Basically the ac function consists of three functions.

- Goto function g
- Failure function f
- Output function o

Goto function g : This function is used to mapping the input text with the pattern which are stored in the form of states. It tells that whether the particular pattern is matched.

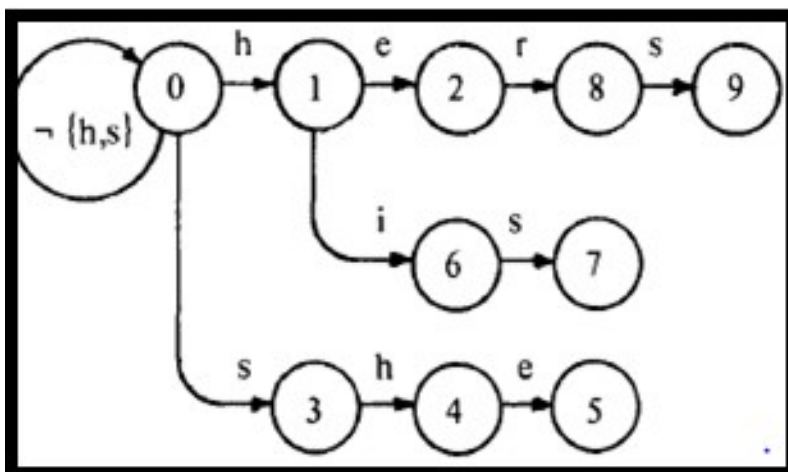


Figure 10: Goto Function

Failure Function f: This function is used to calculate all the failure nodes.

<i>i</i>	1	2	3	4	5	6	7	8	9
<i>f(i)</i>	0	0	0	1	2	0	3	0	3

Figure 11: Failure Function

Output function o: this function provides output of all the valid transition states.

<i>i</i>	<i>output(i)</i>
2	{he}
5	{she, he}
7	{his}
9	{hers}

Figure 12: Output Function

Advanced version of A-C algorithm: PFAC algorithm is advanced parallel version of A-C algorithm. As we know that this algorithm is compatible for GPU. It runs on processor that supports CUDA, including NVIDIA drivers. So we will modify this algorithm to be implemented by using POSIX libraries. Improved A-C algorithm is better than simple A-C algorithm. So this algorithm can be applied to various fields where pattern reorganization is needed. PFAC pattern matching takes running time linear in the sum of length of all anomaly keywords. Failure function is removed in this approach. So no need to store values for failure state. The PFAC algorithm always operates in $O(n)$ running time. In PFAC algorithm a thread considers only the patterns at a particular character. So there is no need to backtrack the state machine. For this reason all the failure transition states are removed. So scope of our algorithm is very big that it can be implemented in many fields like:

- Gene findings
- Virus scanners
- Intrusion detection system

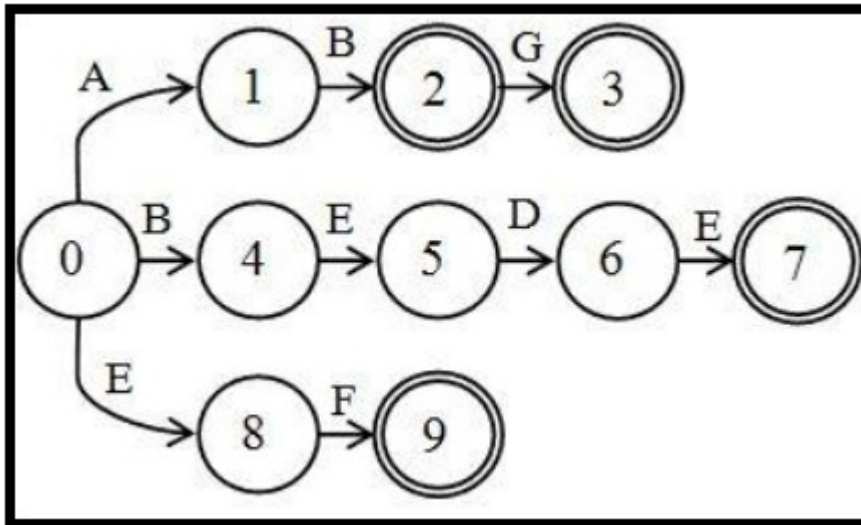


Figure 13: Failure Function of Advanced version of A-C

In this example “AB”, “ABG”, “BEDE”, “EF” is the patterns and ABEDE is input strings. Let us assume that thread tk is assigned to character as shown in Figure13: for traversing failure-less state machine. After traversing the substring AB, thread k reached state no 2 where AB matched pattern is found. But state 2 does not have valid state for character E. So thread tk terminates at this state. Thread k+1 is assigned to input character B. After traversing substring BEDE, thread k+1 reached state 7 where matched pattern BEDE is found.

CHAPTER 4

OBJECTIVE OF STUDY

It is concluded that SQL injection is very serious threat to web applications. After carefully studying different techniques on Prevention and Detection of SQL injection an attack. It is noticed that still more work is to be done in this field. Many researchers contribute different techniques and methods like taint based approach, Static analysis, Dynamic analysis, black box testing, proxy filters, Instruction set randomization work is done in pattern matching concept in SQL injection attacks.

We have concluded that we have to go with pattern matching process for this we have used improved version of A-C algorithm which is also known as Parallel failure-less A-C algorithm. Our objective is to prevent all known types of SQL injection attacks using this algorithm. This advanced version of A-C algorithm can be used in various other applications where large number of patterns has to recognize.

In previous version of A-C algorithm all known types of SQL injection attacks are not handled properly like time based attacks, alternate encoding. So to remove this limitations we have used advanced version of A-C algorithm and one more advantage of this algorithm is that it is a parallel venison of A-C algorithm and in this algorithm failure function is removed so that no backtrack is needed which saves our time. And also memory because no failure function values need to be store. Since pattern matching is a field which can be used for intrusion detection systems, for handling preventing different types of attacks. Prevention and detection of SQL injection attacks can be easily done by pattern matching techniques.

5.1 Formulation of Hypothesis:

After studying various methods for detection and prevention of SQL injection attacks. It is concluded that still more work is need to be done in this field. As SQL injection is listed as top most security threat. Many Researchers provide their different methods in this field. One research group implemented detection and prevention of SQL injection attacks using string matching algorithm but their methodology is not able to detect all types of SQL injection attacks.

A Hypothesis is formulated that if we will use advance version of A-C algorithm then all known SQL injection attacks can be prevented. And this algorithm is parallel version of previous one so performance of advanced version algorithm is better than previous implemented algorithm.

5.2 Research Design

A-C is multiple string pattern matching algorithm. A-C algorithm uses two phases first phase is construction of finite state machine and second phase is searching phase. Previous research group implemented this algorithm to detect and prevent SQL injection attacks. We are using advanced version of A-C algorithm. In our research methodology we focused on two things:-

- Removed failure links A-C algorithm (PFAC) in which we simply remove the failure links so our state machine not needs to backtrack which will save our time and memory.
- We have introduced the concept of thread. In PFAC thread is allocated to each character of the inputted stream and each character can be traversed parallel. In our PFAC approach thread not need to backtrack state machine because the thread consider only the patterns starting at a particular character. Previous implemented algorithm is not able to detect anomaly patterns which are started with some

special characters. So we are removing this limitation in our work to provide better result and to deal with all types of SQL injection attacks.

Architecture for Parallel execution

- A-C algorithm uses two phases first phase is construction of finite state machine and second phase is searching phase. Searching of patterns in the text. In our method Removed failure links in A–C. We divide the pattern set into number of threads and same number of threads will be created. For each pattern set DFA will be created and matched pattern will be generated as an output.

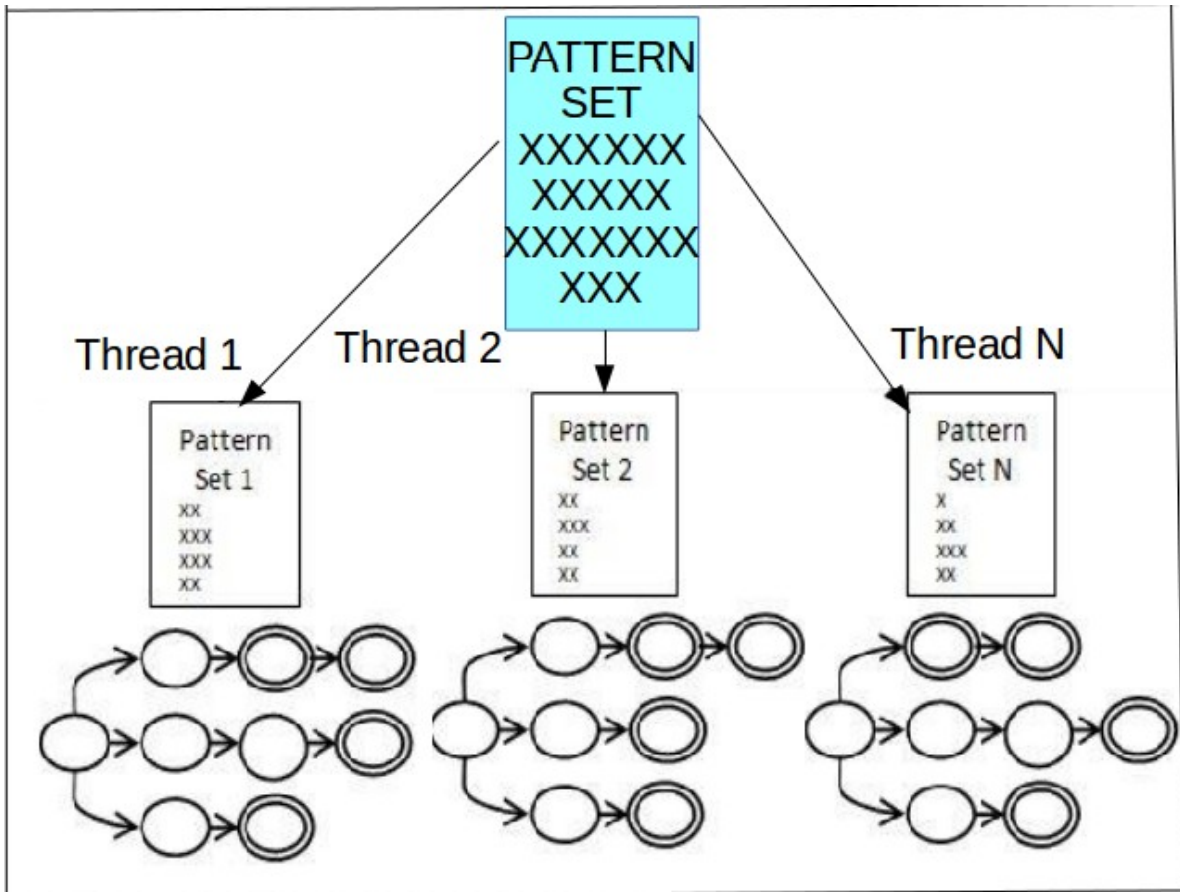


Figure 14: Thread Based Technology

5.2.1 Pseudo code of Anomaly Pattern matching algorithm

Step1: query <- user generated query.

APL [] <- Anomaly pattern list.

Step2: If PFAC (query,query length,APL[][]== null) then

Step3: Patternscore=matchvalue((query,Anomalypattern[j])/patternlength(Anomaly pattern list[j]))* 100

Step4: If (pattern score > Threshold value) then

Step5: Return Alarm to Administrator else

Step6: Return Query Accepted

Step7: End If.

Step8: Else Return Query Rejected

Step9: End if.

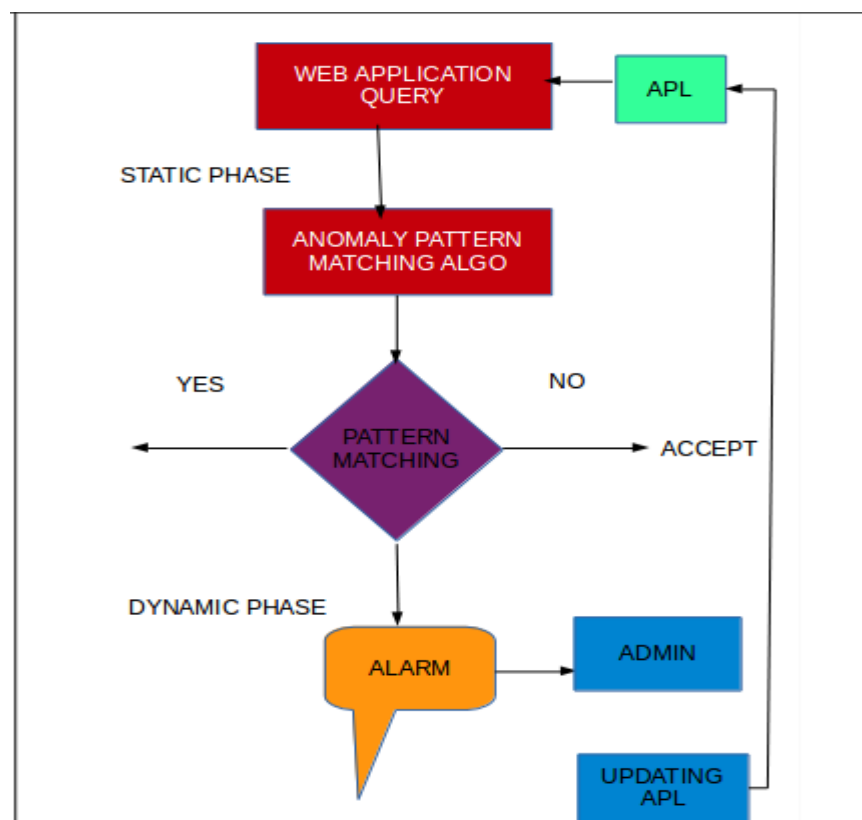


Figure 15: Proposed Architecture

Web Application Query: From where user input data from user form.

Anomaly Pattern algorithm: Here this algorithm is used to calculate the pattern score of the user query.

Pattern score will be the maximum score of that query. If pattern score comes out to be greater than threshold value. Then the query is vulnerable and it is send to the administrator.

APL (ANOMALY PATTERN LIST): It is a collection of all vulnerable patterns. If some new kind of pattern is generated then it will add to APL by the administrator.

5.2.2 Calculation of Pattern score

The proposed scheme will calculate the pattern score value of the user input SQL query. If the query is exactly match 100% then the query is injected. Otherwise, the maximum matching score is calculated. Threshold value (40%). Threshold value is calculated by Reverse Engineering.

5.2.3 PFAC Algorithm

In PFAC implementation, all failure transition are removed from the state machine however, in the PFAC algorithm, a thread considers only the patterns starting at a particular character and therefore the threads of the PFAC do not need to back-track state machine. In pre-computation phase of the algorithm automation is constructed using the set of keywords scanning the SQL query statement reading every character in SQL query exactly once and taking constant time for each read of character. Pseudo code of algorithm is given below.

Step1: Start

Step2: Build DFA of the given patterns.

Step3: $n \leftarrow$ SQL query string length.

Step4: $m \leftarrow$ minimum SQL pattern lengths.

Step5: begin $(n-m+1)$ threads for parallel processing.

Step6: for each thread do

Step7: Thread[i] start scanning from i and go to DFA of Patterns. Range of thread index is

0 to n-m.

Step8: If Failure state than stop & exit threads.

Step9: Else If final states reach in the DFA Than

(i)Maintain the occurrences.

(ii)Continue scanning character and make transition table in the DFA.

(iii) Go to step 8.

Step10: Else

(i)Continue scanning character and make transition table in the DFA.

(ii)Go to step 8.

5.3 Software Requirement Specification

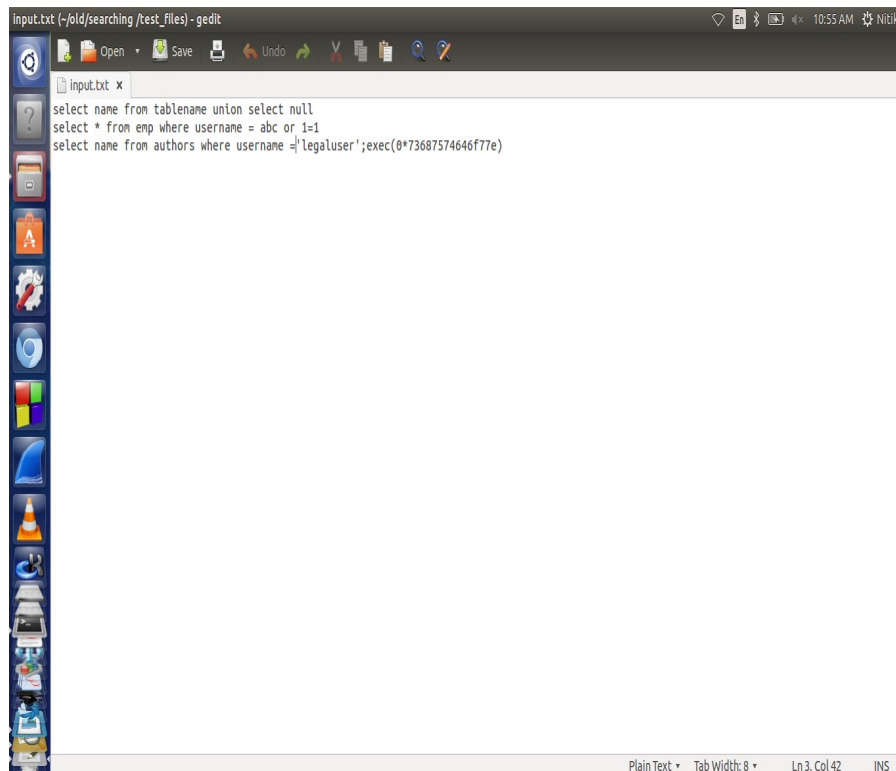
Following are the specific software and hardware requirements for the project to work effectively.

- For parallelism we use Pthread library.
- C programming language.
- GNU/Linux Platform.

CHAPTER 6 RESULTS AND DISCUSSIONS

Many approaches are there to prevent and detect SQL injection attacks. SQL injection is a top most security threat to web applications. All type of SQL injection attacks are detected and prevented by our approach. The important aspect of this technique is that, it does not generate the false positive results. We have tested our methodology on all types of SQL injection attacks and compared it with previous one. Thus it is concluded that our methodology is able to detect and prevent various types of SQL injection attacks. Snapshots for our work are given below: -

Old work

A screenshot of a text editor window titled 'input.txt (-/old/searching /test_files) - gedit'. The window contains three lines of SQL injection queries: 'select name from tablename union select null', 'select * from emp where username = abc or 1=1', and 'select name from authors where username = 'legaluser';exec(0*73687574646f77e)'. The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 3, Col 42', and 'INS'.

```
input.txt (-/old/searching /test_files) - gedit
input.txt x
select name from tablename union select null
select * from emp where username = abc or 1=1
select name from authors where username = 'legaluser';exec(0*73687574646f77e)
Plain Text • Tab Width: 8 • Ln 3, Col 42  INS
```

Figure 16: Shows Input queries file

We can check from input query file that three types of vulnerable queries are entered

1. Union query
2. Tautology
3. Alternate encoding

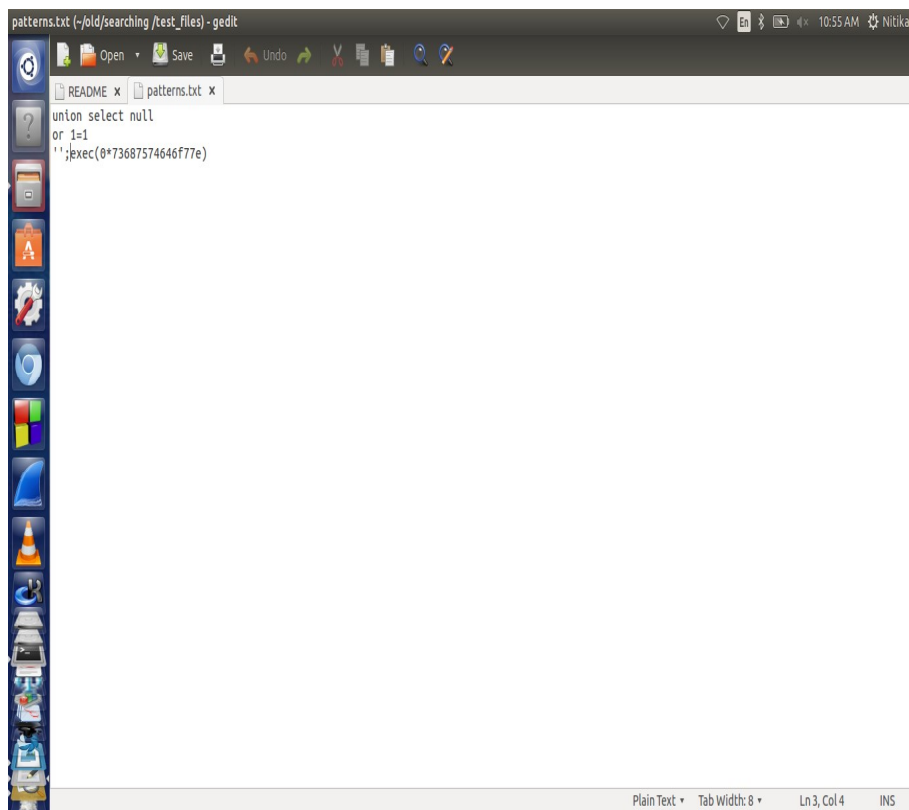


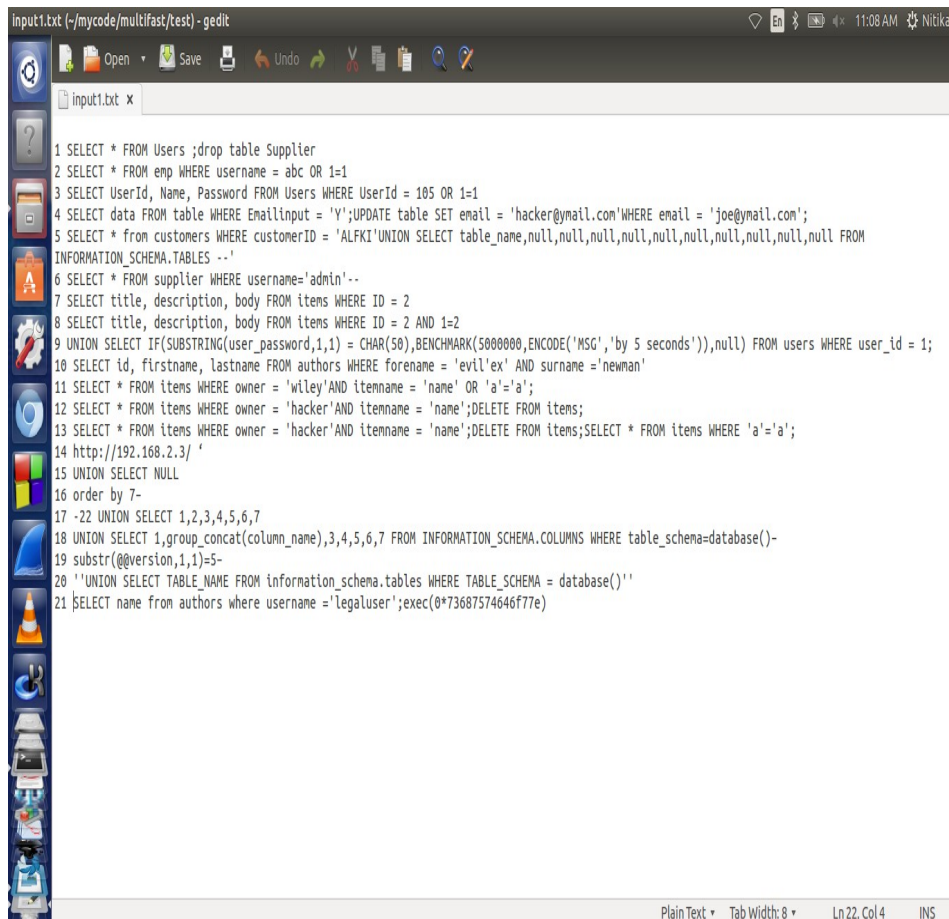
Figure 17: Stored anomaly patterns for user anomaly queries

```
nitika@nitika-HP-430-Notebook-PC:~/old/multifast
nitika@nitika-HP-430-Notebook-PC:~$ cd old
nitika@nitika-HP-430-Notebook-PC:~/old$ cd ahocorasick
nitika@nitika-HP-430-Notebook-PC:~/old/ahocorasick$ make
make: 'libahocorasick-1.3.a' is up to date.
nitika@nitika-HP-430-Notebook-PC:~/old/ahocorasick$ cd
nitika@nitika-HP-430-Notebook-PC:~$ cd old
nitika@nitika-HP-430-Notebook-PC:~/old$ cd multifast
nitika@nitika-HP-430-Notebook-PC:~/old/multifast$ make
make: multifast is up to date.
nitika@nitika-HP-430-Notebook-PC:~/old/multifast$ ./multifast -n i -p test_files/patterns.txt -i test_files/input.txt -o nfs
1      28 union select null
2      85 or 1=1
nitika@nitika-HP-430-Notebook-PC:~/old/multifast$
```

Figure 18: Shows anomaly patterns for user input queries

Figure 18 shows that only two types of attacks are handled from three types of attacks. That are Union queries, and tautology attacks but alternate encoding attack is not handled.

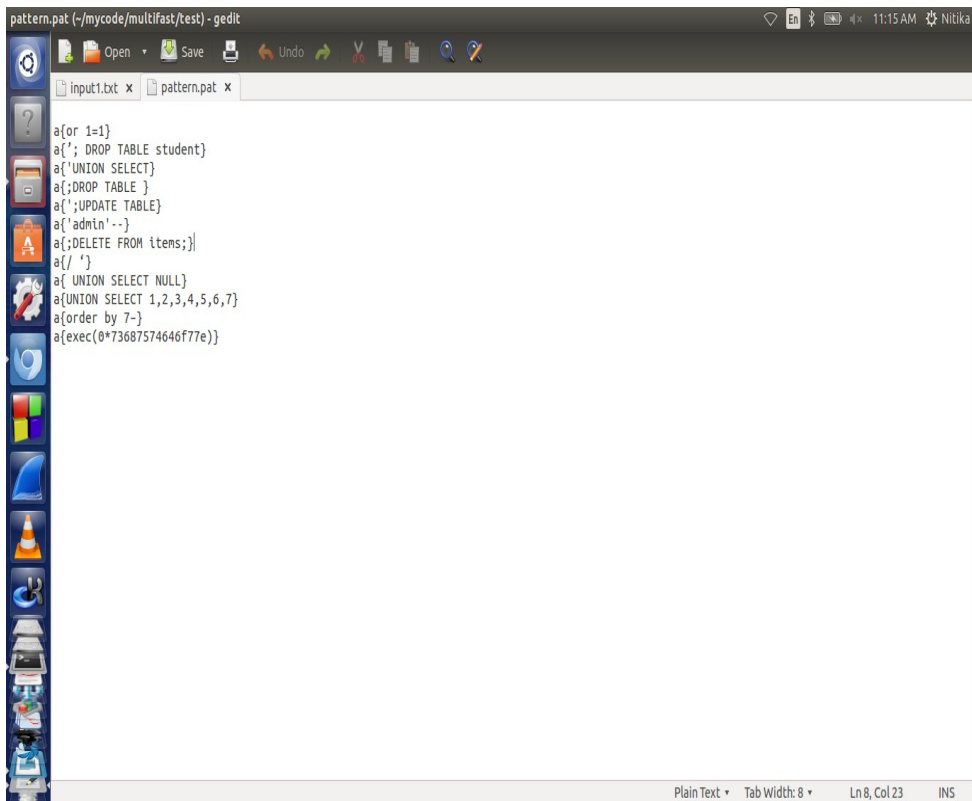
Present work



```
input1.txt (-/mycode/multifast/test) - gedit
1 SELECT * FROM Users ;drop table Supplier
2 SELECT * FROM emp WHERE username = abc OR 1=1
3 SELECT UserId, Name, Password FROM Users WHERE UserId = 105 OR 1=1
4 SELECT data FROM table WHERE Emailinput = 'Y';UPDATE table SET email = 'hacker@gmail.com'WHERE email = 'joe@gmail.com';
5 SELECT * from customers WHERE customerID = 'ALFKI'UNION SELECT table_name,null,null,null,null,null,null,null,null,null FROM
INFORMATION_SCHEMA.TABLES --'
6 SELECT * FROM supplier WHERE username='admin'--
7 SELECT title, description, body FROM items WHERE ID = 2
8 SELECT title, description, body FROM items WHERE ID = 2 AND 1=2
9 UNION SELECT IF(SUBSTRING(user_password,1,1) = CHAR(50),BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')),null) FROM users WHERE user_id = 1;
10 SELECT id, firstname, lastname FROM authors WHERE forename = 'evil'ex' AND surname = 'newman'
11 SELECT * FROM items WHERE owner = 'wiley'AND itemname = 'name' OR 'a'='a';
12 SELECT * FROM items WHERE owner = 'hacker'AND itemname = 'name';DELETE FROM items;
13 SELECT * FROM items WHERE owner = 'hacker'AND itemname = 'name';DELETE FROM items;SELECT * FROM items WHERE 'a'='a';
14 http://192.168.2.3/ '
15 UNION SELECT NULL
16 order by 7-
17 -22 UNION SELECT 1,2,3,4,5,6,7
18 UNION SELECT 1,group_concat(column_name),3,4,5,6,7 FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schena=database()-
19 substr(@@version,1,1)=5-
20 ''UNION SELECT TABLE_NAME FROM information_schema.tables WHERE TABLE_SCHEMA = database()''
21 |SELECT name from authors where username = 'Legaluser';exec(0*73687574646f77e)
```

Figure 19: Shows user input queries

This figure contains all types of known SQL injection queries and it also contains legitimate queries.



```
pattern.pat (-/mycode/multifast/test) - gedit
input1.txt x pattern.pat x
a{or 1=1}
a{' ; DROP TABLE student}
a{' UNION SELECT}
a{' ; DROP TABLE }
a{' ; UPDATE TABLE}
a{' admin'--}
a{' ; DELETE FROM items;}
a{/ ' }
a{' UNION SELECT NULL}
a{' UNION SELECT 1,2,3,4,5,6,7}
a{' order by 7-}
a{' exec(0*73687574646f77e)}
```

Figure 20: Shows anomaly patterns related to user input queries

All known SQL injection patterns are stored in APL (Anomaly Pattern List).


```

nitika@nitika-HP-430-Notebook-PC: ~/mycode/multifast
nitika@nitika-HP-430-Notebook-PC:~$ cd mycode
nitika@nitika-HP-430-Notebook-PC:~/mycode$ cd ahocorasick
nitika@nitika-HP-430-Notebook-PC:~/mycode/ahocorasick$ make
mkdir -p build
cc -o build/ahocorasick.o -c ahocorasick.c -Wall
cc -o build/node.o -c node.c -Wall
ar -cvq build/libahocorasick.a build/ahocorasick.o build/node.o
a - build/ahocorasick.o
a - build/node.o
nitika@nitika-HP-430-Notebook-PC:~/mycode/ahocorasick$ cd
nitika@nitika-HP-430-Notebook-PC:~$ cd mycode
nitika@nitika-HP-430-Notebook-PC:~/mycode$ cd multifast
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast$ make
mkdir -p build
cc -o build/multifast.o -c multifast.c -Wall -I../ahocorasick
cc -o build/pattern.o -c pattern.c -Wall -I../ahocorasick
cc -o build/reader.o -c reader.c -Wall -I../ahocorasick
cc -o build/strmm.o -c strmm.c -Wall -I../ahocorasick
cc -o build/walker.o -c walker.c -Wall -I../ahocorasick
cc -o build/multifast build/multifast.o build/pattern.o build/reader.o build/str
mm.o build/walker.o -Wall -L../ahocorasick/build -lahocorasick
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast$ build/multifast -P test/pattern.pat -drxp test/input1.txt
test/input1.txt: @338 @00000152 p000003 {'UNION SELECT}
test/input1.txt: @488 @000001E8 p000006 {'admin'-}
test/input1.txt: @1004 @000003EC p000007 {;DELETE FROM items;}
test/input1.txt: @1090 @00000442 p000007 {;DELETE FROM items;}
test/input1.txt: @1165 @00000480 p000008 {2f 20 e2 80 98}
test/input1.txt: @1173 @00000495 p000009 { UNION SELECT NULL}
test/input1.txt: @1195 @000004AB p000011 {6f 72 64 65 72 20 62 79 20 37 e2 80 93}
test/input1.txt: @1216 @000004C0 p000010 {UNION SELECT 1,2,3,4,5,6,7}
test/input1.txt: @1396 @00000574 p000003 {'UNION SELECT}
test/input1.txt: @1542 @00000606 p000012 {exec(0*73687574646f77e)}
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast$

```

Figure 21: Shows matched anomaly patterns

Command build/multifast -P test/pattern.pat -drxp test/input1.txt

- P specifies pattern file
- n show match number
- d show start position (decimal)
- x show start position (hex)
- r show representative string for the pattern
- p show pattern

```
nitika@nitika-HP-430-Notebook-PC: ~/mycode/multifast
nitika@nitika-HP-430-Notebook-PC:~$ cd mycode
nitika@nitika-HP-430-Notebook-PC:~/mycode$ cd ahocorasick
nitika@nitika-HP-430-Notebook-PC:~/mycode/ahocorasick$ make
mkdir -p build
cc -o build/ahocorasick.o -c ahocorasick.c -Wall
cc -o build/node.o -c node.c -Wall
ar -cvq build/libahocorasick.a build/ahocorasick.o build/node.o
a - build/ahocorasick.o
a - build/node.o
nitika@nitika-HP-430-Notebook-PC:~/mycode/ahocorasick$ cd
nitika@nitika-HP-430-Notebook-PC:~$ cd mycode
nitika@nitika-HP-430-Notebook-PC:~/mycode$ cd multifast
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast$ make
mkdir -p build
cc -o build/multifast.o -c multifast.c -Wall -I../ahocorasick
cc -o build/pattern.o -c pattern.c -Wall -I../ahocorasick
cc -o build/reader.o -c reader.c -Wall -I../ahocorasick
cc -o build/strmm.o -c strmm.c -Wall -I../ahocorasick
cc -o build/walker.o -c walker.c -Wall -I../ahocorasick
cc -o build/multifast build/multifast.o build/pattern.o build/reader.o build/strmm.o build/walker.o -Wall -L../ahocorasick/build -lahocorasick
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast$ cat test/input1.txt | ./build/multifast -P test/pattern.pat -dp -
@338 {'UNION SELECT'}
@488 {'admin'--}
@1004 {;DELETE FROM items;}
@1090 {;DELETE FROM items;}
@1165 {2f 20 e2 80 98}
@1173 { UNION SELECT NULL}
@1195 {6f 72 64 65 72 20 62 79 20 37 e2 80 93}
@1216 {'UNION SELECT 1,2,3,4,5,6,7'}
@1396 {'UNION SELECT'}
@1542 {exec(0*73687574646f77e)}
```

Figure 22: Shows input anomaly patterns

```
nitika@nitika-HP-430-Notebook-PC: ~/mycode/multifast/build
mkdir -p build
cc -o build/multifast.o -c multifast.c -Wall -I../ahocorasick
cc -o build/pattern.o -c pattern.c -Wall -I../ahocorasick
cc -o build/reader.o -c reader.c -Wall -I../ahocorasick
cc -o build/strmm.o -c strmm.c -Wall -I../ahocorasick
cc -o build/walker.o -c walker.c -Wall -I../ahocorasick
cc -o build/multifast build/multifast.o build/pattern.o build/reader.o build/strmm.o build/walker.o -Wall -L../ahocorasick/build -lahocorasick
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast$ ./multifast -P ../test/pattern.pat -nxrpiv ../test/input1.txt
bash: ./multifast: No such file or directory
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast$ cd build
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast/build$ ./multifast -P ../test/pattern.pat -nxrpiv ../test/input1.txt
Loading Patterns From '../test/pattern.pat'
Added successfully: p000001 - {or 1=1}
Added successfully: p000002 - {e2 80 99 3b 20 64 72 6f 70 20 74 61 62 6c 65 20 73 74 75 64 65 6e 74}
Added successfully: p000003 - {'union select}
Added successfully: p000004 - {;drop table }
Added successfully: p000005 - {';update table}
Added successfully: p000006 - {'admin'--}
Added successfully: p000007 - {;delete from items;}
Added successfully: p000008 - {2f 20 e2 80 98}
Added successfully: p000009 - { union select null}
Added successfully: p000010 - {union select 1,2,3,4,5,6,7}
Added successfully: p000011 - {6f 72 64 65 72 20 62 79 20 37 e2 80 93}
Added successfully: p000012 - {exec(0*73687574646f77e)}
Total Patterns: 12
Searching 1 files
../test/input1.txt: #1 @0000018 p000004 {;drop table }
../test/input1.txt: #2 @0000057 p000001 {or 1=1}
../test/input1.txt: #3 @000009c p000001 {or 1=1}
../test/input1.txt: #4 @00000d3 p000005 {';update table}
../test/input1.txt: #5 @0000152 p000003 {'union select}
../test/input1.txt: #6 @00001E8 p000006 {'admin'--}
../test/input1.txt: #7 @00003EC p000007 {;delete from items;}
../test/input1.txt: #8 @0000442 p000007 {;delete from items;}
../test/input1.txt: #9 @000048D p000008 {2f 20 e2 80 98}
../test/input1.txt: #10 @0000495 p000009 { union select null}
../test/input1.txt: #11 @00004AB p000011 {6f 72 64 65 72 20 62 79 20 37 e2 80 93}
../test/input1.txt: #12 @00004C0 p000010 {union select 1,2,3,4,5,6,7}
../test/input1.txt: #13 @0000574 p000003 {'union select}
../test/input1.txt: #14 @0000606 p000012 {exec(0*73687574646f77e)}
nitika@nitika-HP-430-Notebook-PC:~/mycode/multifast/build$
```

Figure 23: Shows successfully added patterns

S.NO	QUERY LENGTH	PATTERN LENGTH	PATTERN SCORE(%)
1	40	20	76.19
2	51	5	42.5
3	65	8	49.5
4	65	10	61.9
5	80	13	99.4
6	30	25	71.4
7	50	9	42.85
8	70	7	46.6
9	90	6	51.4
10	90	9	77.14
11	80	10	76.19
12	50	12	57.14
13	50	20	95.2
14	50	18	85.71
15	40	12	45.7
16	50	11	52.38
17	20	23	65.7
18	50	10	47.6
19	60	7	40

Table 1: Calculation of Pattern Score for vulnerable queries

If Pattern score is greater than Threshold value (40%) then it is vulnerable query return to administrator. length of total pattern=1050.

Schemes	tautology	Illegal/incorrect queries	Union queries	Piggy-backed queries	Stored procedures	inference	Alternate encoding
AMNESIA	✓	✓	✓	✗	✓	✓	✓
SQLrand	✓	✗	✓	✗	✓	✓	✓
Candid	✓	✗	✗	✗	✗	✗	✗
SQLGuard	✓	✗	✗	✗	✗	✗	✗
Negative tainting	✓	✓	✓	✗	✓	✓	✓
Pattern matching	✓	✓	✓	✓	✓	✗	✗
Proposed method	✓	✓	✓	✓	✓	✓	✓

Table 2: Comparison of our scheme with other scheme

We have done comparison of our scheme with previous implemented scheme and it shows that our scheme is able to detect and prevent all types of SQL injection attacks.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

As we know that demand of web application increases day by day. All off-line works are converting into on-line work. Threat to web application is also increasing day by day. SQL injection is listed top most security threat to web application many researchers performed research on this and provides many different approaches to deal with this attack. Many on-line tools are also available for the detection and prevention of SQL injection attacks. Our approach consist of identifying vulnerable queries using anomaly pattern matching algorithm to find out the SQL injection attacks, inserting newly identified SQL injection attacks to improve accuracy of the system. Preventing SQL injection attacks using anomaly pattern matching algorithm is designed to work well with all databases. It could be implemented and tested to work with other databases such as MySQL, Oracle, PostgreSQL, Fox Pro, and Microsoft Access.

CHAPTER 8 REFERENCES

1. AS.Gadgikar, “PREVENTING SQL INJECTION ATTACKS USING NEGATIVE TAINTING APPROACH”, 2013 IEEE.
2. Debabrata Kar, Suvasini Panigrahi, “PREVENTION OF SQL INJECTION ATTACK USING QUERY TRANSFORMATION AND HASHING”, 2012.
3. Dr. MA.Amutha Prabakar, M.KarthiKeyan, Prof.K.Marimuthu, “AN EFFICIENT TECHNIQUE FOR PREVENTING SQL INJECTION ATTACK USING PATTERN MATCHING ALGORITHM”, 2013 IEEE.
4. G.Aghila, R.Ezumalai, “COMBINATORIAL METHOD FOR PREVENTING SQL INJECTION ATTACKS”, 2009.
5. Jeom Kim, “INJECTION ATTACK DETECTION USING THE REMOVAL OF SQL QUERY ATTRIBUTE VALUES”, 2011 IEEE.
6. Michelle Ruse, Tanmoysarkar, samik basu, “SQL INJECTION DETECTION VIA AUTOMATIC TEST CASE GENERATION OF PROGRAMS”, 2010.
7. Ntagwabira Lambert, Kang Song Lin, “USE OF QUERY TOKENIZATION TO DETECT AND PREVENT SQL INJECTION ATTACKS”, 2010 IEEE.
8. PrithviBisht, Sruthi Bandhakavi, P.Madhusudan, V.N.Venkatakrishanan, “CANDID: PREVENTING SQL INJECTION ATTACKS USING DYNAMIC CANDIDATE EVALUATIONS”.
9. Srinivas Avireddy, Varalaksmi Perumal, Narayan Gowraj, Ram Srivastva Kannan, Prashanth Thinakaran, “AN APPLICATION SPECIFIC RANDOMIZED ENCRYPTION ALGORITHM TO PREVENT SQL INJECTION”.
10. Stephen W. Boyd, Angelos D. Keromytis, “SQLRAND PREVENTION OS

SQLINJECTION ATTACKS”.

11. Shelly Rohilla, Pradeep kumar mittal, “DATABASE SECURITY BY PREVENTING SQL INJECTION ATTACKS IN STORED PROCEDURE”, 2013.
12. William G.J Halfond, Alessandro Orso, “AMNESIA: ANALYSIS AND MONITORING FOR NEUTRALIZING SQL INJECTION ATTACKS”, 2005.