National College of Computer Studies

Institute of Science and Technology

Tribhuvan University

Final Year Project

on

SQL Injection Detection And Analysis

Submitted By:

Sushant Shrestha(3009/70)

Sakroj Karki(3002/70)

Sujan Khatri(3008/70)

Madav Prasad Chaurasia(2986/70)

Under the Supervision of

Mr. Dilli Prasad Sharma

June, 2017

NCCS College

NATIONAL COLLEGE OF COMPUTER STUDIES

Tribhuvan University

SUPERVISOR’S RECOMMENDATION

I hereby recommend that this project prepared under my supervision by Mr. Dilli Prasad Sharma entitled **“SQL injection Detection And Analysis”** in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology be processed for the evaluation.

…………………………………………

Mr. Dilli Prasad Sharma

Institute of Science and Technology

Tribhuvan University

ABSTRACT

This documentation on “SQL injection detection and analysis” overviews on SQL injections and their detection. Here, detection is done through static analysis and through hash algorithm implementation. This system is concerned with the security. This system is designed so that it blocks many of the SQL injections. In addition to that, analysis is done on time complexity of static and hash algorithms to detect SQL injections.

ACKNOWLEDGEMENT

Our society comprises of people supporting each other, as there is only much that one can do and so much more that can be done by many.

This project has been the honest hard work of many people. Firstly, we would like to express our gratitude to our sir Mr.Dilli Prasad Sharma” who has been there to support us in every step of the way. And guiding us with his valuable advice for the project. And secondly, we are also thankful to Department of Computer Science, National College of Computer Studies, for approving our project and giving us an opportunity to enter the field of Computer security through this project.

Last but not the least, I would like to thank out friends, mentor and the internet for providing us with valuable information for the completion of the project.

Table of Contents

Chapter I: Introduction

1.1 Background

There are thousands of security breaches that take place every day. According to Bagchi [1] 75% of the firms websites and web applications were vulnerable to the Internet Security Breaches. SQL injection is one of them. A SQL injection can be prevented using extensive coding and defensive coding. A SQL injection technique is regarding the defensive coding that is used to detect the SQL attacks or injections. Our aim is to provide a comparative study of detection techniques that have been developed till now. With our study we will be able to differentiate between the different parameters and the reader will be able to make an analysis on the different parameters that he or she may require in their study. Comparative analysis will be able to help readers select between the techniques that the suit with the attack they are handling.

1.2 Problem Definition**:**

SQL injection is a worldwide web intrusion problem. SQL injection has a wide range of attacks as well as is prune to develop viewing the current usage of the SQL servers. Attacks like SQL injection, Cross Side Scripting and many more can be used to get the data from the database. SQL injection has led to a lot of damage of data or leakage of data in the past years.

Among many of the insecurities, our project is concerned with the SQL injection. Our application stands between the server and the user request to analyze the incoming request for detection and prevention of SQL injection. Furthermore this project is designated to use multiple algorithms to determine the most suitable detection and prevention of SQL injection.

Our project will remove the SQL injection attacks, so our project will solve the following:

- SQL attacks
- Suitable technique while handling a SQL attack

1.3 Objective:

SQL injection detection and prevention has been a worldwide matter since the development of SQL in 1970s. The first public discussions of SQL injection started appearing around 1998 [2] and since then many SQL injections have been found and dealt with. Since then many SQL injection techniques have been developed.Our main Objective is to use SQL injection detection technique and compare them in accordance with the most feasible parameters. So, our project consists of two main goals:

- To implement SQL Injection Detection techniques
- To compare the detection techniques on the basis of time.

1.4 Process model:

The most popular and simple type of model waterfall process model was suitable for our project. Its distinctive feature is step by step process from requirements analysis to maintenance. The figure below shows the step by step process of the waterfall model.

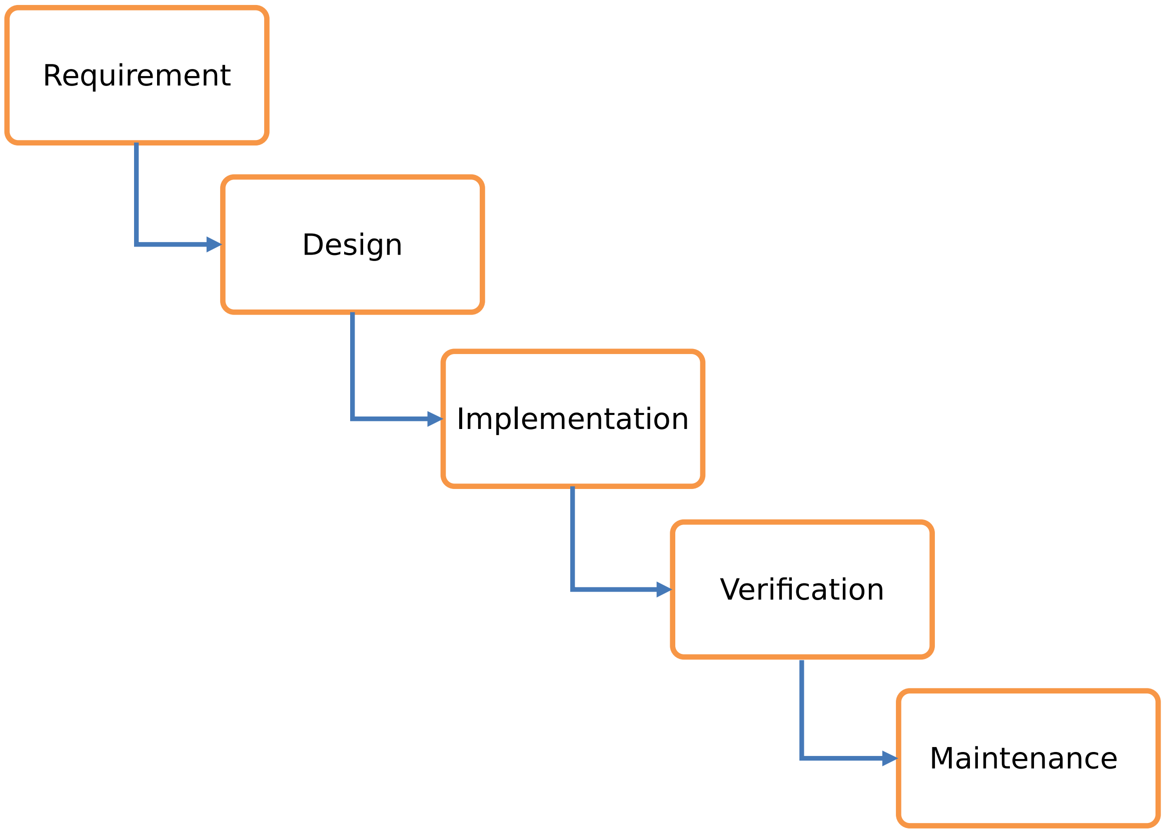


Figure 1.4 : Showing Waterfall Process Model

CHAPTER II: RESEARCH METHODOLOGY

2.1 Literature Review

There have been many application built for SQL intrusion detection and prevention. Some are:

1. DevBug
2. Gamja

DevBug is a basic PHP Static Code Analysis (SCA) tool written mostly in JavaScript. The idea behind DevBug is to make basic PHP Static Code Analysis accessible online, to raise security awareness and to integrate SCA into the development process. DevBug could be used to quickly test a page of PHP that you think may have some potential vulnerabilities, to run across a piece of code you have found on Google that you are unsure of or to directly write your own code in.

Gamja will find XSS (Cross site scripting) & SQL Injection weak point also URL parameter validation error.

2.2 Tools Used

The tools that are used in this project:

- PHP
- HTML
- CSS
- MySQL server
- Apache server

2.3 Algorithm Development/Used

2.3.1 Static Algorithm

We have devised our own static algorithm in this project. Here, the static algorithm is a program specifically developed for the purpose of detection and prevention of SQL injections.

We can divide it ito four main parts:

- Quote detection:

  What this part does is, it checks if the input given by the user is free of quote or not. As we know single quote and double quote can be used to partition the database (single quote ‘ , double quote ”) which makes the database hackable.

- Logical part:

  Here, in this part of our static algorithm we can find the code to detect logical inputs i.e. this parts deals with the logical operation such as AND and OR. And prevents the user form going any further in the database.

- Bad input detection:

  As we know a simple sql injection contains sql quries which can be used to get the information from the database. So, what this part does is it prevents the user to get database information using sql quries as sql quires are rendered as bad inputs.

- Allowing only good inputs:

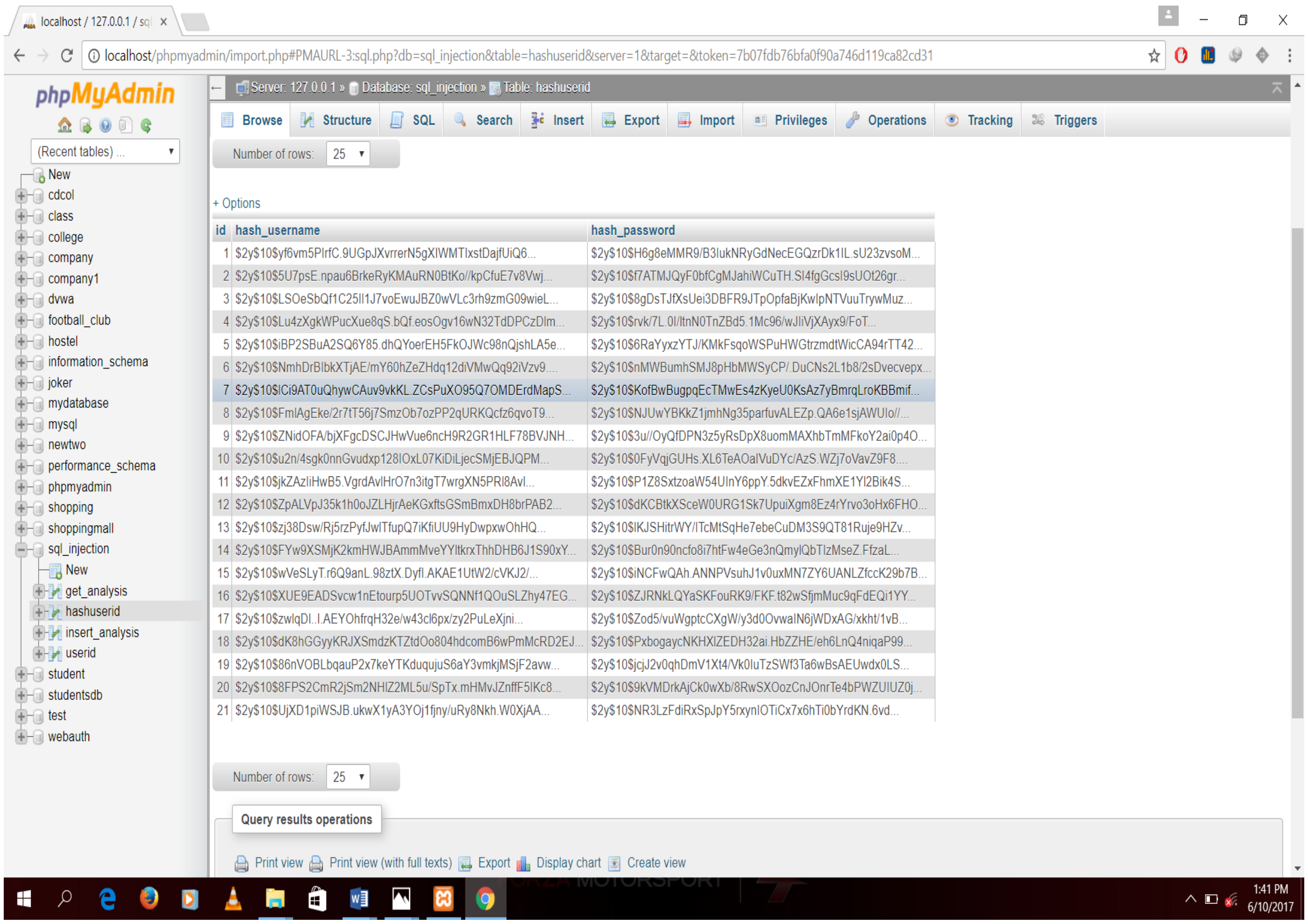
  Inputs having sql quries, AND, OR, quote can is considered as bad inputs as these are mainly used to hack the database. Here, in this part of the static algorithm ,the algorithm checks if the input given is free of the above bad inputs and checks whether the input given is authentic or not.

2.3.2 Hash Algorithm

In this algorithm we can find hash value use to authenticate the user. In the proposed approach there is a need for one extra column in database, which contains the EX-OR of the Hash values of username and password at the time, when a user account is created for the first time and stores in the database. Whenever user wants to login to database his/her identity is checked using user name and password and its hash values. These hash values are calculated at runtime using store procedure when user wants to login into the database.

During the authentication of user, the SQL query with hash parameters is used. Hence, if a user tries the injection to the query and our proposed methodology is working with SQL query, it will automatically find out the injections as the potentially harmful content and rejects the values.

Therefore, it cannot bypass the authentication process. The advantage of the proposed technique is that the hackers do not know about the hash values of user name and password. So, it is impossible for the hacker to bypass the authentication process through the general SQL injection techniques. The SQL injection attacks can only be done on codes which are entered through user entry form but the hash values are calculated at run time at backend before creating SELECT query to the underlying database therefore the hacker cannot calculate the hash values as it dynamic at Runtime.



localhost/phpmyadmin/import.php#PMAURL-3:sql.php?db=sql_injection&table=hashuserid&server=1&target=&token=7b07fdb76bfa0f90a746d119ca82cd31

Server: 127.0.0.1 » Database: sql_injection » Table: hashuserid

id	hash_username	hash_password
1	$2y$10$yf6vm5PlrfC.9UGpJXvrrerN5gXIWMTlxstDajfUiQ6...	$2y$10$H6g8eMMR9/B3lukNRyGdNecEGQzrDk1IL.sU23zvsoM...
2	$2y$10$5U7psE.npau6BrkeRyKMAuRN0BtKo//kpCfuE7v8Vwj...	$2y$10$f7ATMJQyF0bfCgMJahiWCuTH.Sl4fgGcsl9sUOt26gr...
3	$2y$10$LSOeSbQf1C25lI1J7voEwuJBZ0wVLc3rh9zmG09wieL...	$2y$10$8gDsTJfXsUei3DBFR9JTpOpfaBjKwlpNTVuuTrywMuz...
4	$2y$10$Lu4zXgkWPucXue8qS.bQf.eosOgv16wN32TdDPCzDlm...	$2y$10$rvk/7L.0l/ltnN0TnZBd5.1Mc96/wJliVjXAyx9/FoT...
5	$2y$10$iBP2SBuA2SQ6Y85.dhQYoerEH5FkOJWc98nQjshLA5e...	$2y$10$6RaYyxzYTJ/KMkFsqoWSPuHWGtrzmdtWicCA94rTT42...
6	$2y$10$NmhDrBlbkXTjAE/mY60hZeZHdq12diVMwQq92iVzv9....	$2y$10$nMWBumhSMJ8pHbMWSyCP/.DuCNs2L1b8/2sDvecvepx...
7	$2y$10$lCi9AT0uQhywCAuv9vkKL.ZCsPuXO95Q7OMDErdMapS...	$2y$10$KofBwBugpqEcTMwEs4zKyeU0KsAz7yBmrqLroKBBmif...
8	$2y$10$FmlAgEke/2r7tT56j7SmzOb7ozPP2qURKQcfz6qvoT9...	$2y$10$NJUwYBKkZ1jmhNg35parfuvALEZp.QA6e1sjAWUlo//...
9	$2y$10$ZNidOFA/bjXFgcDSCJHwVue6ncH9R2GR1HLF78BVJNH...	$2y$10$3u//OyQfDPN3z5yRsDpX8uomMAXhbTmMFkoY2ai0p4O...
10	$2y$10$u2n/4sgk0nnGvudxp128IOxL07KiDiLjecSMjEBJQPM...	$2y$10$0FyVqjGUHs.XL6TeAOalVuDYc/AzS.WZj7oVavZ9F8....
11	$2y$10$jkZAzliHwB5.VgrdAvlHrO7n3itgT7wrgXN5PRl8Avl...	$2y$10$P1Z8SxtzoaW54UlnY6ppY.5dkvEZxFhmXE1Yl2Bik4S...
12	$2y$10$ZpALVpJ35k1h0oJZLHjrAeKGxftsGSmBmxDH8brPAB2...	$2y$10$dKCBtkXSceW0URG1Sk7UpuiXgm8Ez4rYrvo3oHx6FHO...
13	$2y$10$zj38Dsw/Rj5rzPyfJwlTfupQ7iKfiUU9HyDwpxwOhHQ...	$2y$10$lKJSHitrWY/lTcMtSqHe7ebeCuDM3S9QT81Ruje9HZv...
14	$2y$10$FYw9XSMjK2kmHWJBAmmMveYYltkrxThhDHB6J1S90xY...	$2y$10$Bur0n90ncfo8i7htFw4eGe3nQmylQbTlzMseZ.FfzaL...
15	$2y$10$wVeSLyT.r6Q9anL.98ztX.Dyfl.AKAE1UtW2/cVKJ2/...	$2y$10$iNCFwQAh.ANNPVsuhJ1v0uxMN7ZY6UANLZfccK29b7B...
16	$2y$10$XUE9EADSvcw1nEtourp5UOTvvSQNNf1QOuSLZhy47EG...	$2y$10$ZJRNkLQYaSKFouRK9/FKF.t82wSfjmMuc9qFdEQi1YY...
17	$2y$10$zwlqDl..l.AEYOhfrqH32e/w43cl6px/zy2PuLeXjni...	$2y$10$Zod5/vuWgptcCXgW/y3d0OvwalN6jWDxAG/xkht/1vB...
18	$2y$10$dK8hGGyyKRJXSmdzKTZtdOo804hdcomB6wPmMcRD2EJ...	$2y$10$PxbogaycNKHXlZEDH32ai.HbZZHE/eh6LnQ4niqaP99...
19	$2y$10$86nVOBLbqauP2x7keYTKduqujuS6aY3vmkjMSjF2avw...	$2y$10$jcjJ2v0qhDmV1Xt4/Vk0luTzSWf3Ta6wBsAEUwdx0LS...
20	$2y$10$8FPS2CmR2jSm2NHlZ2ML5u/SpTx.mHMvJZnffF5lKc8...	$2y$10$9kVMDrkAjCk0wXb/8RwSXOozCnJOnrTe4bPWZUlUZ0j...
21	$2y$10$UjXD1piWSJB.ukwX1yA3YOj1fjny/uRy8Nkh.W0XjAA...	$2y$10$NR3LzFdiRxSpJpY5rxynlOTiCx7x6hTi0bYrdKN.6vd...

Figure 2.4.2: Showing hash value implementing hash algorithm

CHAPTER III: SYSTEM ANAYSIS

3.1 Requirement Specification

3.1.1 Functional Requirements

Functional requirement can be defined as the main function that the system is supposed to have. Like the functional requirement of a pen is to write, of a camera is to take pictures, of a phone is to make calls, etc. Like this the functional requirement defines the main aspect of the system as to “What it is supposed to do?”.

The functional requirement from our project are listed as below:

1. Allows only authenticated users to use the database.
2. Prevents unauthorized access.
3. Prevents data leakage from the database.
4. Uses various algorithms (static algorithm and hash algorithm) to prevent from sql injection attacks.

3.1.2 Non Functional requirements

Non Functional requirements can be defined as the functions extraneous to the system i.e. a system can have extra features such as a phone can now also be use for accessing media, using internet ,reading text files and many more and not only calls. Thus, non functional requirements are like the extra ability that the system possesses then what it was built to do.

The non functional requirements from our project are listed below:

1. Gives the time complexity of each algorithm used.
2. Helps to compare which algorithms will be faster to execute.
3. Comparative study can be done.

3.2 Feasibility Study

3.2.1 Economic Feasibility

The purpose of economic feasibility is to have an analysis of economic benefit to the organization. Since our system isn't used in our daily life and the equipment we need are those that we already have, so the system is economically feasible.

3.2.2 Technical Feasibility

The SQL injection detection system can be developed using easily available tools and can be handled in a home environment. Each of the equipment that we ought to use are affordable and are mostly free of cost. Hence we can conclude that the system is technically feasible.

3.2.3 Operational Feasibility

The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with existing business environment. Our application and study creates a stable, reliable, maintainable, supportable, usable, sustainable and affordable application and analysis thus we can say that the system is operationally feasible.

3.2.4 Schedule Feasibility

Our Schedule is feasible as we have completed our assignments as per the time. i.e. the application development in four months has proceeded as scheduled.

3.3 Use-case diagram

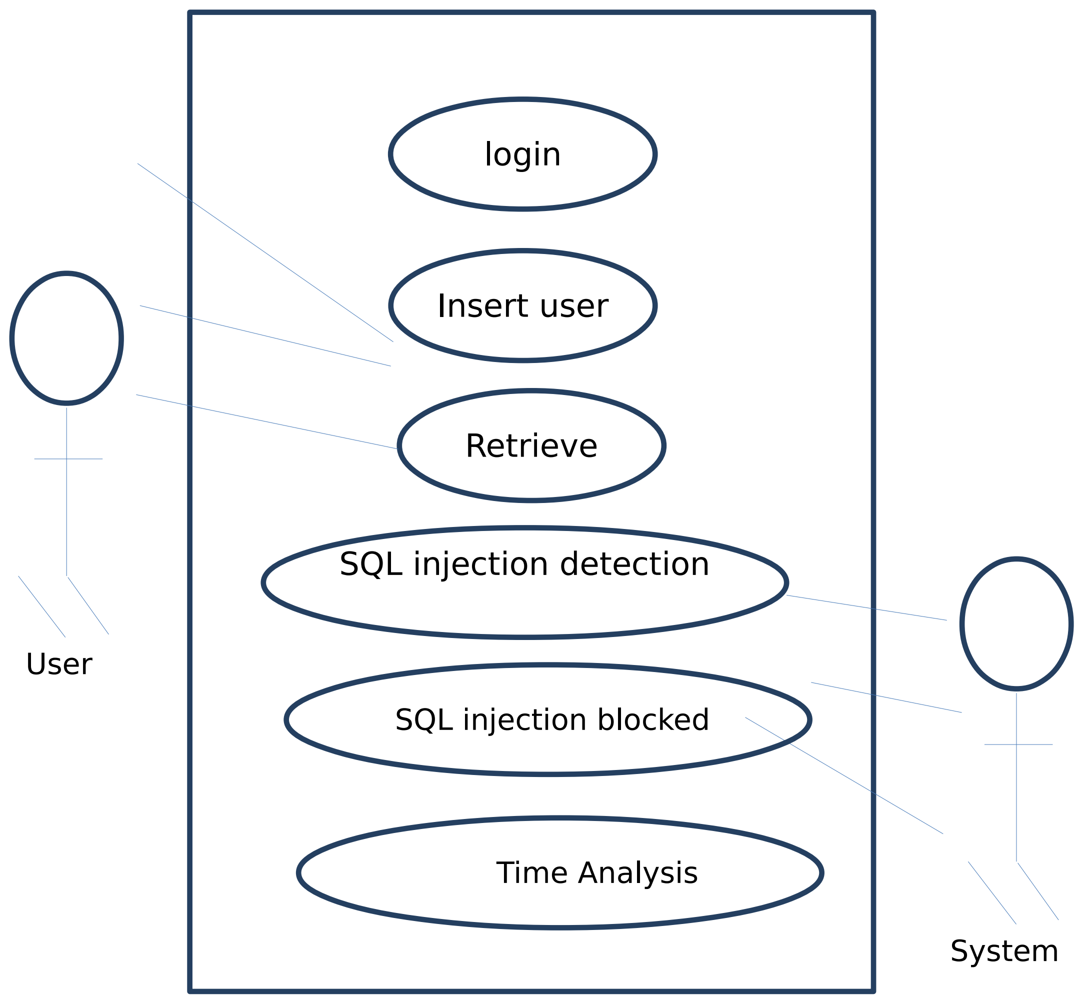


Figure 3.3: Showing use case diagram of the system

The above diagram is a use case diagram showing the structure of our application. Here, the user can login, then insert new users or retrieve information on the existing users. Now on doing those things our system checks the user behavior to the system. As in our system firstly checks weather the input is an sql injection or not. If it is an injection then it blocks that request by the user and does not allow the user to get any more information on the database. Lastly, our system uses two algorithm to block the sql injection so we have included a time analysis system to compare the time taken by the algorithms to finish their jobs.

3.4 Sequence Diagram

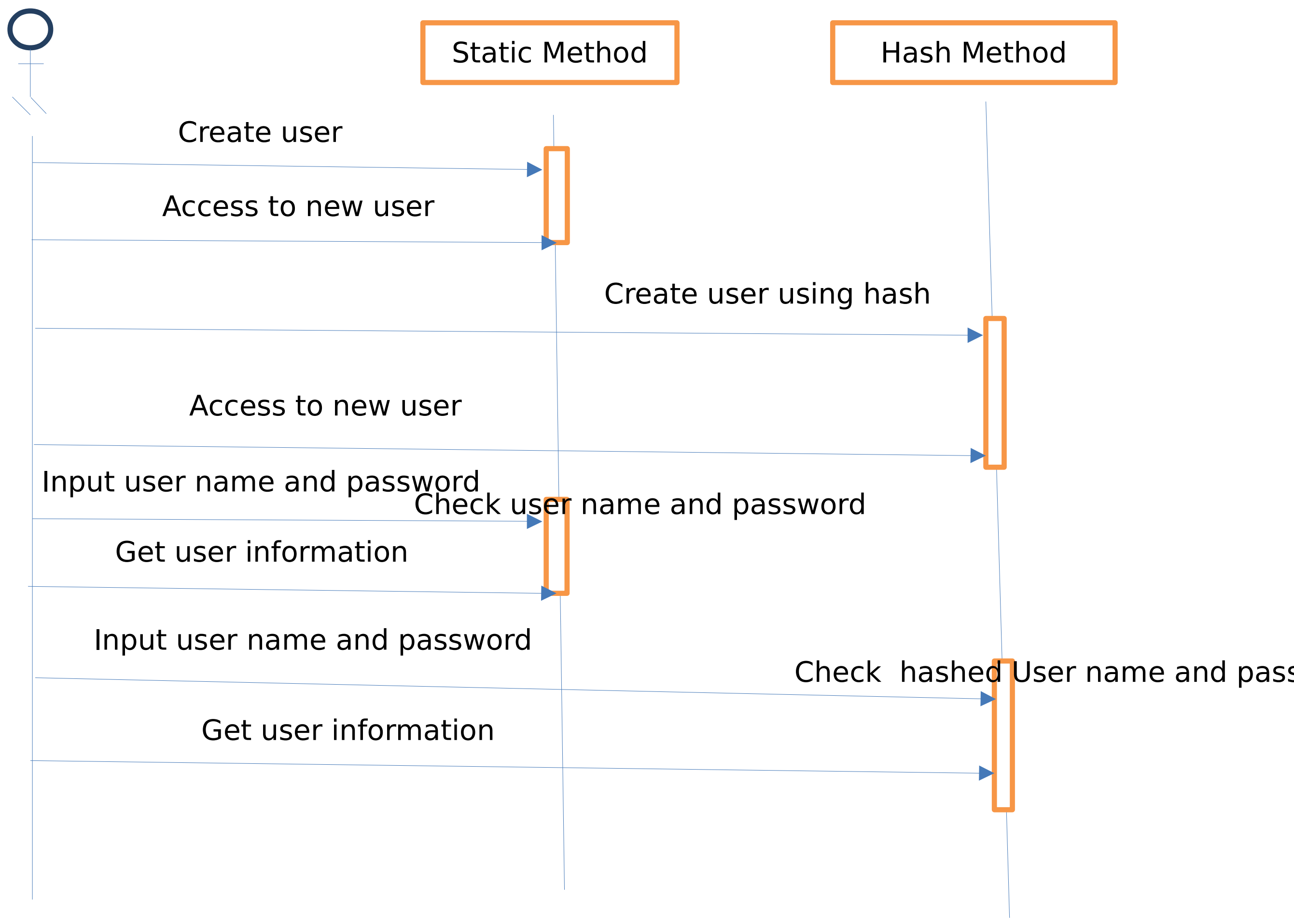


Figure 3.4: Showing Sequence Diagram of the system

3.5 Class diagram

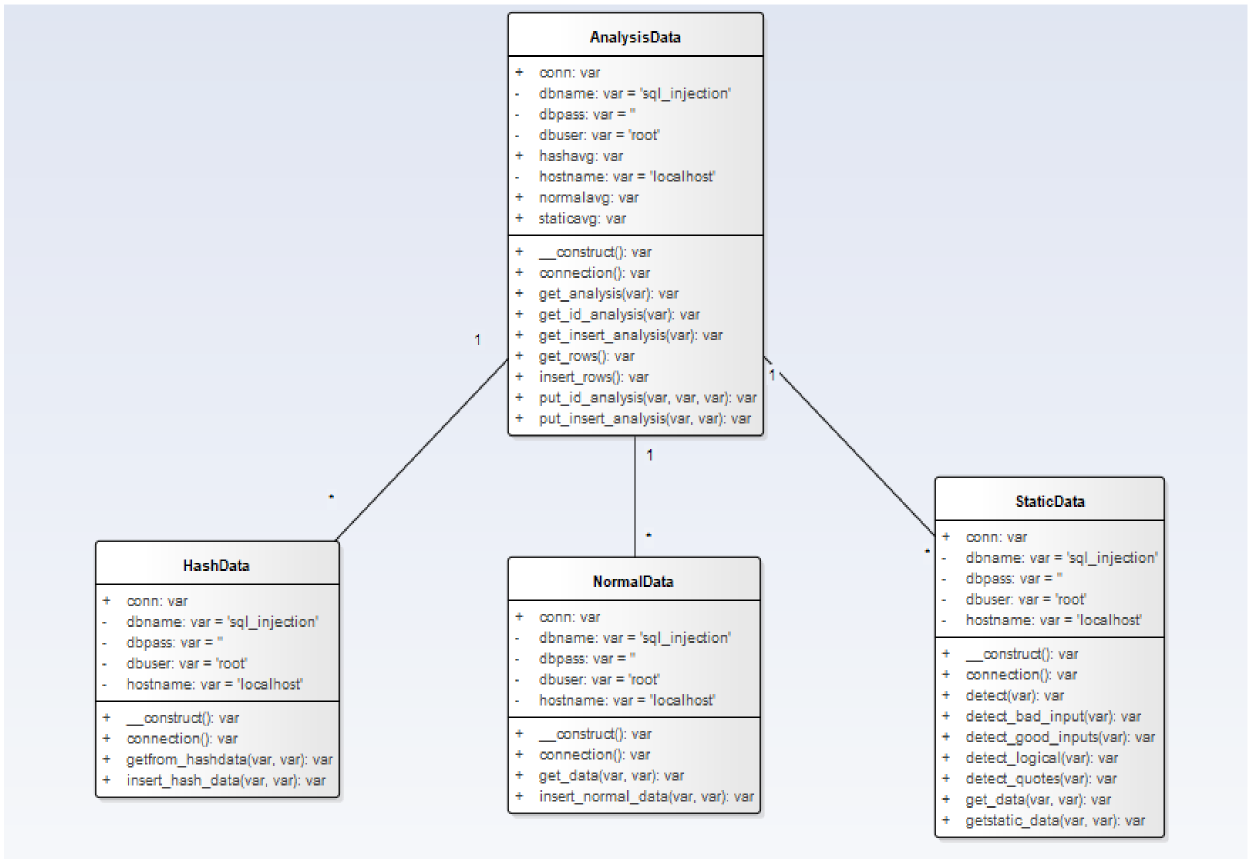

AnalysisData
+ conn: var
- dbname: var = 'sql_injection'
- dbpass: var = ''
- dbuser: var = 'root'
+ hashavg: var
- hostname: var = 'localhost'
+ normalavg: var
+ staticavg: var
+ __construct(): var
+ connection(): var
+ get_analysis(var): var
+ get_id_analysis(var): var
+ get_insert_analysis(var): var
+ get_rows(): var
+ insert_rows(): var
+ put_id_analysis(var, var, var): var
+ put_insert_analysis(var, var): var
1
1
1
*
*
*
HashData
+ conn: var
- dbname: var = 'sql_injection'
- dbpass: var = ''
- dbuser: var = 'root'
- hostname: var = 'localhost'
+ __construct(): var
+ connection(): var
+ getfrom_hashdata(var, var): var
+ insert_hash_data(var, var): var
NormalData
+ conn: var
- dbname: var = 'sql_injection'
- dbpass: var = ''
- dbuser: var = 'root'
- hostname: var = 'localhost'
+ __construct(): var
+ connection(): var
+ get_data(var, var): var
+ insert_normal_data(var, var): var
StaticData
+ conn: var
- dbname: var = 'sql_injection'
- dbpass: var = ''
- dbuser: var = 'root'
- hostname: var = 'localhost'
+ __construct(): var
+ connection(): var
+ detect(var): var
+ detect_bad_input(var): var
+ detect_good_inputs(var): var
+ detect_logical(var): var
+ detect_quotes(var): var
+ get_data(var, var): var
+ getstatic_data(var, var): var

Figure 3.5: Showing Class diagram of the system

CHAPTER IV: SYSTEM DESIGN

4.1 System architecture

The figure below explains how our basic system architecture should be. Here, the user injects sql in for of a request to get the information on the database of the server. Our web application handles those request before passing them through to the server. Here, the request is scanned for sql injection. If our system finds the request to be sql injection then it responses to the request by a error message. But if the request is a correct one then only the request is sent to the server and the correct response is given to the client/user. Thus this is how our system basically works.

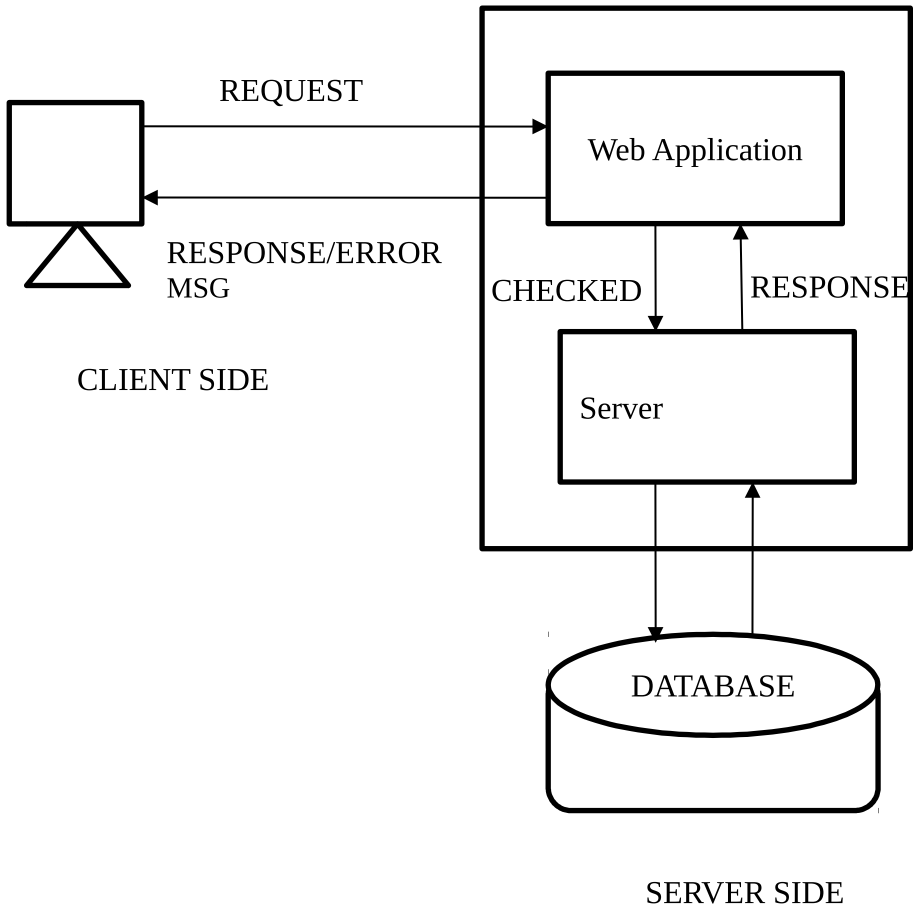


Figure 4.1: Showing System architecture

4.2 Program flow diagram

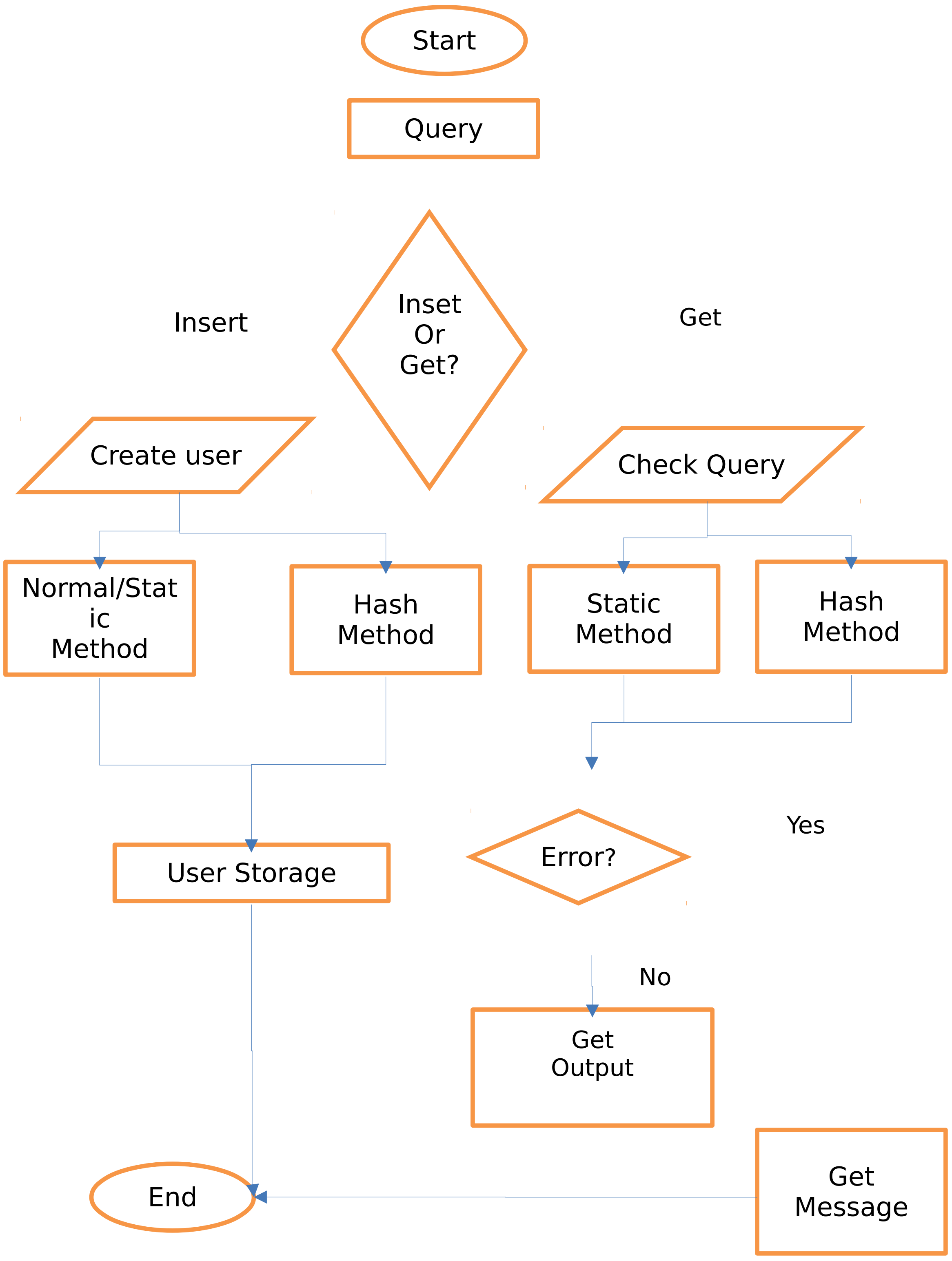


Figure 4.3: Showing flow of the program

The above diagram shows the actual flow of our program. As we start the program the user input is taken as a query. If that query is for insert user then a user is created in two ways one is by using the normal/static method and the other is by using hash method. Using static method a user is inserted normally whereas using hash method generates a hash username and hash password value of the created user. Then the user is stored in the database. Now to retrieve or get that user our program firstly checks the query. This is also done in two ways hash and static method. Here the hash and static algorithms are used to check the query. If the query does not meet the required standard for the program then an error message is outputted otherwise we will get the required/correct output message.

4.3 User Interface Design

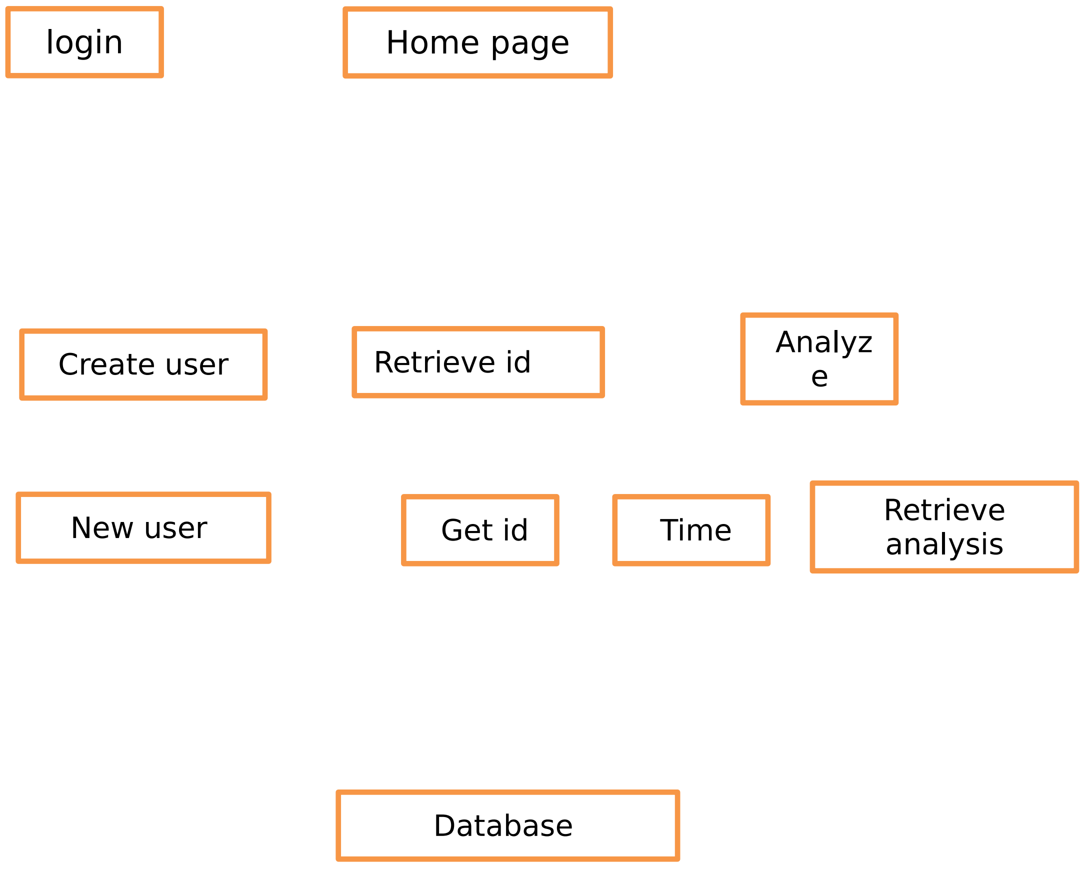


Figure 4.4: Showing user interface design

CHAPTER V: SYSTEM IMPLEMENTATION

5.1 Coding

The code phase of our project is mainly divided into two main parts. One is coding for hash algorithm and other one is coding for static algorithm. The following shows the detailed description of these two parts hash and static algorithms.

5.1.1 Hash coding

5.1.1 Algorithm for hash

For creating user

Step 1: get user name and password

Step 2: convert username and password into hash bits

Step 3: store the username and password into database

For verification:

Step 1: get the username and password

Step 2: get the hash username and password from the database

Step 3: using a loop verify if the hashed username and password match from any one of the database

Step 4: return the required output from the verification

The pseudo code for the above algorithm is give below:

createuser()

```
{
    get (username, password)
    $username = hash(username)
    $password = hash(password)
    database_store($username, $password)
}

getuser()

{
    get (login_username, login_password)
    foreach(hashusername in database)
    $hashusername = get(hashusername)
    $HASHPASSWORD = GET(HASHPASSWORD)

    IF                    (PASSWORD_VERIFY(LOGIN_USERNAME,
$HASHUSERNAME)&&PASSWORD_VERIFY(LOGIN_PASSWORD, $HASHPASSWORD))
}
```

5.2 Static coding

5.2.1. Algorithm

```
Step1: get the query
Step2: check the query for sql injected lines
Step3: if query contains quotes then sql injection detected else goto step 8
Step4: if query contains any sql query words the sql injection is detected else goto step 8
Step5: if query contains any logical operators then sql injection is detected else goto step 8
Step6: if query contains any letter except designated letters then possible sql injection is detected else goto step 8
Step7: pass the query to the database for processing
Step8: display output or error
```

The pseudo code for the above algorithm is shown below:

```
public function detect(query)
{
    if (detect_quotes)
            return "quotes detected"
    if (bad_inputs_detected)
            return "bad inputs"
    if (logical_inputs)
            return "logical inputs detected"
    if(!good_inputs)
```

```
              return "bad inputs "
}

detect_quotes(query)
{
    if (query matches ' or ")
              return true
    else
              return false
}

bad_inputs_detect(query)
{
    bad_inputs = "union", "select", "update", "insert", "delete", "--", "drop", "="
    if (query matches bad_inputs )
              return true
    else
              return false
}

logical_inputs(query)
```

```
{
    if (query matches ("or ", "and"))
            return true
    else
            return false
}

good_inputs(query)
{
    good = "A-Z" or "a-z" or "0-9"
    if (query has only good)
            return true
    else
            return false
}
```

5.3 Testing Strategies

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. This testing phase is also done to verify and validate our system EVENT TRACKING SYSTEM and also to check for the requirements that were proposed, are met or not. For this, many testing strategies has been performed.

White Box Testing

White box testing is a method of testing software that tests internal structures and working of an application. This includes testing each unit modules of Event Tracking System and integration testing of each module to guide system.

Unit Testing

During initial development of the project, each and every unit of the system was tested for any anomalies and errors. Every unit was checked for both syntactical as well as logical error first-hand by the programmer during the coding of the units itself.

Integration Testing

Multiple correct units may not always result in proper, error free system. Once the various blocks were combined to make a single large block, those blocks were again tested to see if they were integrated properly.

Such test was performed by entering the arbitrary data into the system. The errors found cause of improper integration were consequently removed or corrected as necessary.

B**lack Box Testing**

Black box testing is a method of testing software in which internal structure, design implementation of the software is not known to the tester. This includes System testing and functional Testing.

System Testing

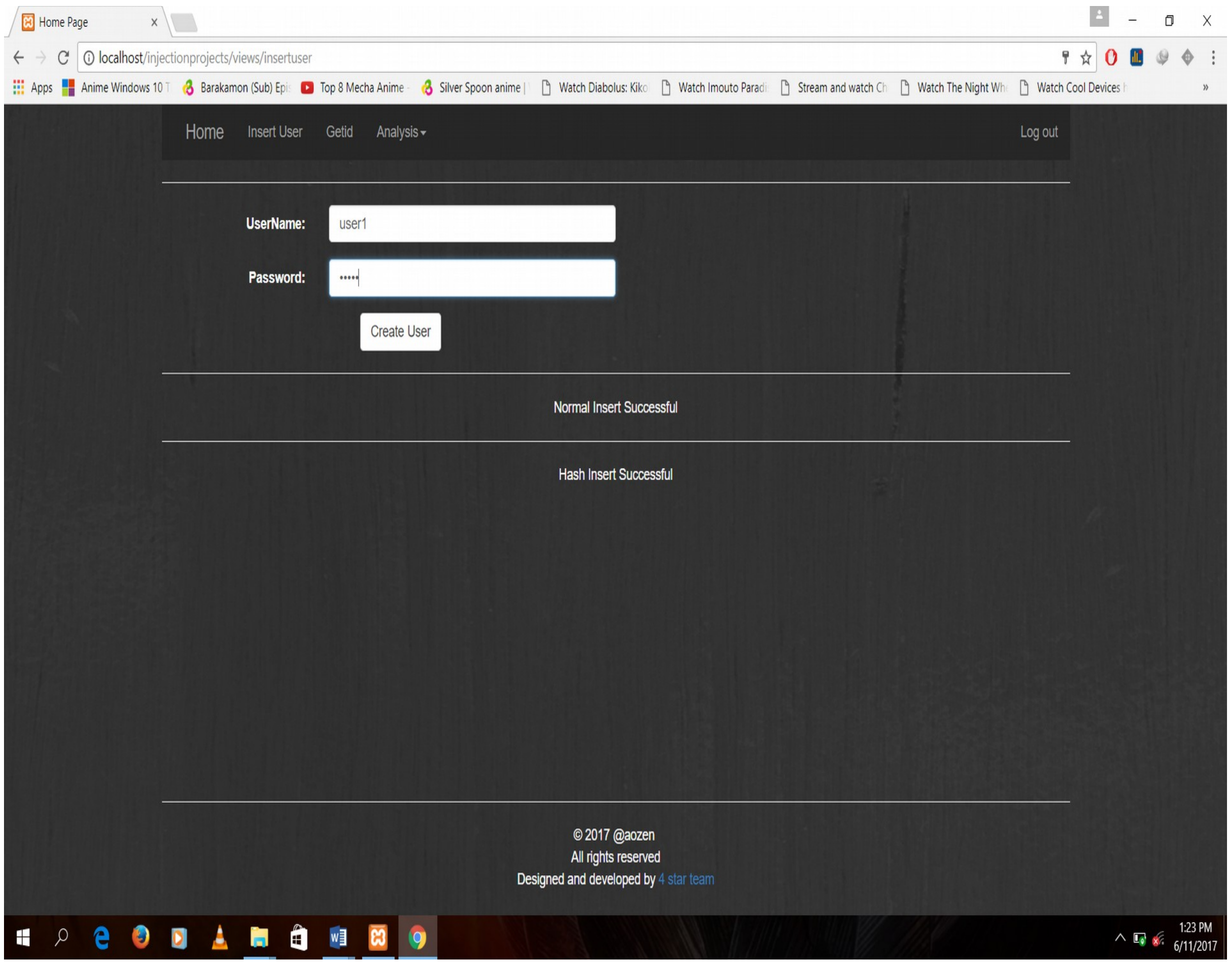
The system was tested across various versions. During the running of the system across various versions, any errors found were subsequently corrected. Finally, this process was repeated until very minimal to no performance lag or error was found.

Functional Testing

To check the functionality of the system, testing was carried to check whether the functions of the system were implemented properly or not. The functions like inserting data, deleting data, retrieving data, etc.

were performed. The output’s deviation from the actually expected output was observed and corrected necessarily.

5.4 Executing Snapshots



UserName: user1

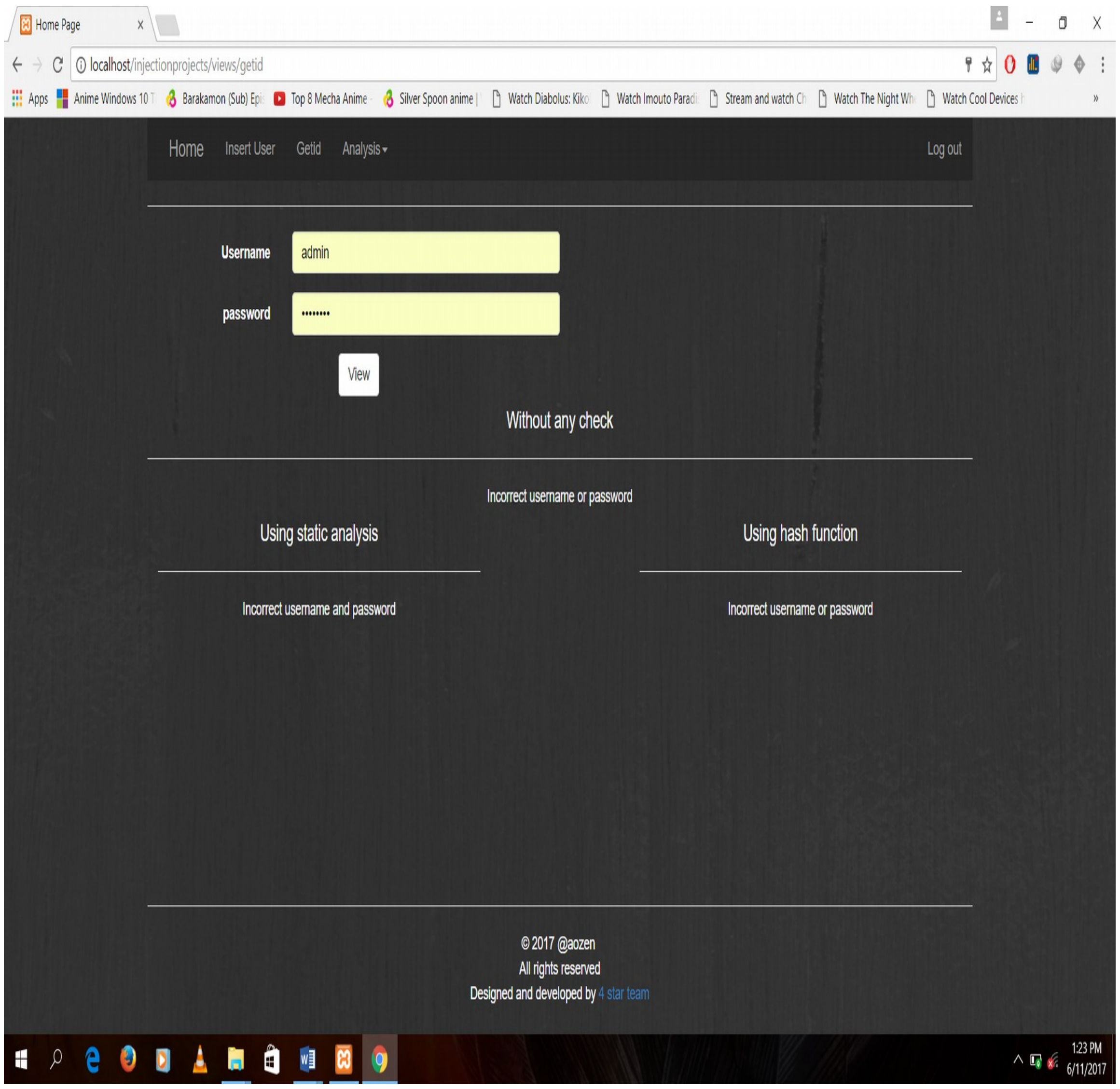
Password: •••••

Normal Insert Successful

Hash Insert Successful

© 2017 @aozen
All rights reserved
Designed and developed by 4 star team

Figure 5.4(a): Creating normal and hash user



Username admin

password ••••••••

Without any check

Incorrect username or password

Using static analysis

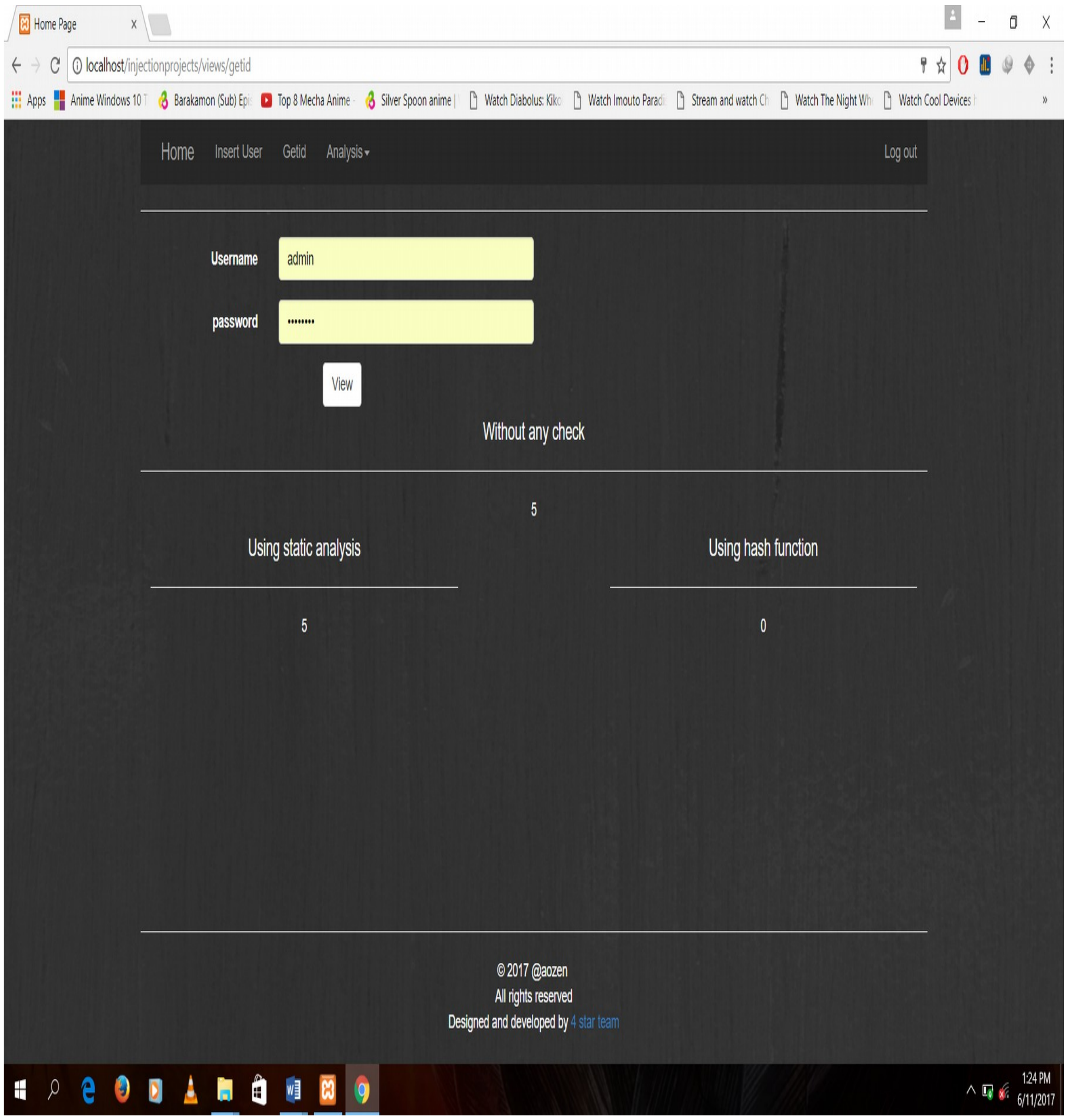
Incorrect username and password

Using hash function

Incorrect username or password

© 2017 @aozen
All rights reserved
Designed and developed by 4 star team

Figure 5.4(b): Incorrect user name and password inserted



Username admin

password ••••••••

Without any check

5

Using static analysis

5

Using hash function

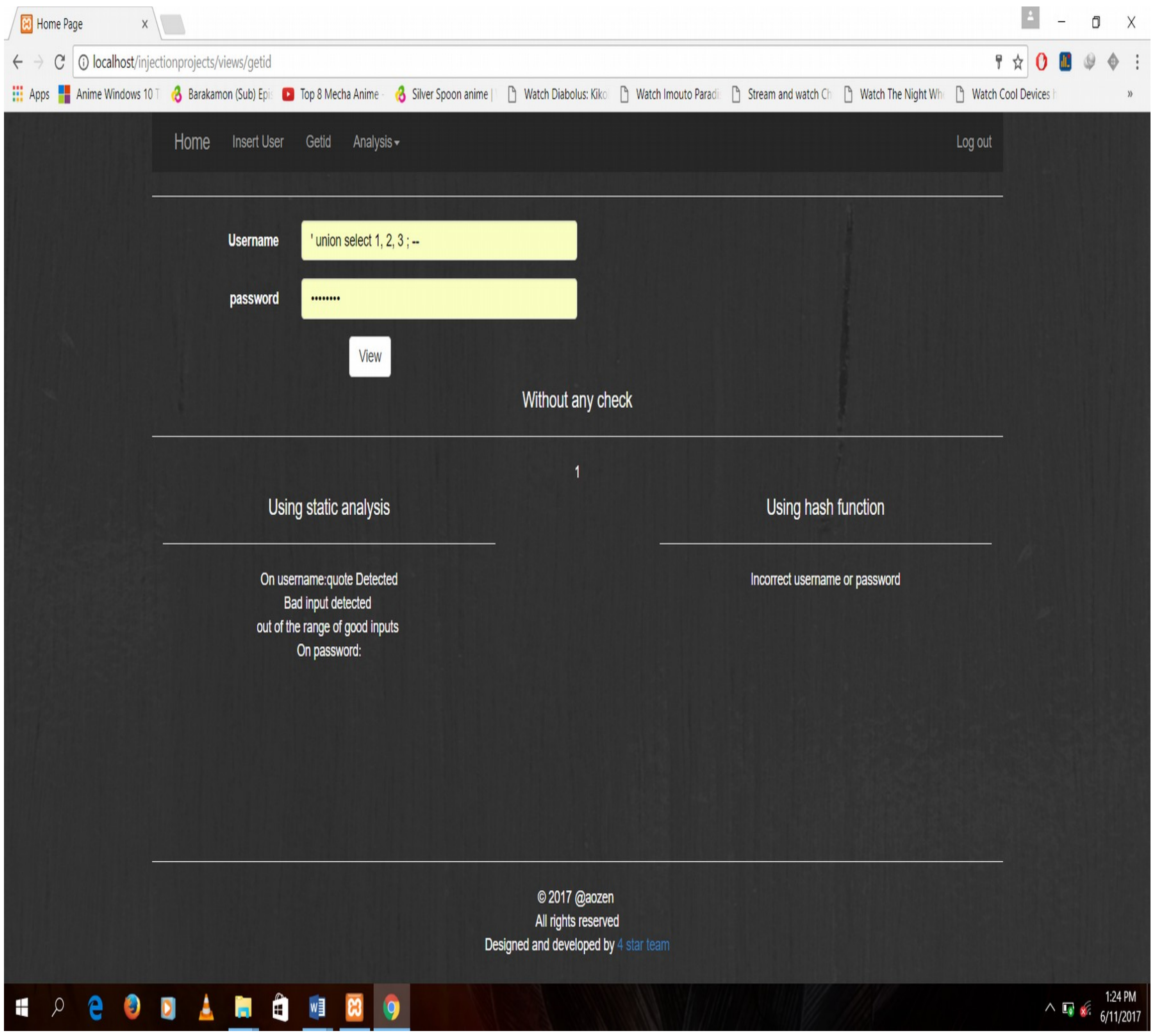
0

© 2017 @aozen

All rights reserved

Designed and developed by 4 star team

Figure 5.4(c): Receiving data from correct inputs



Username: ' union select 1, 2, 3 ; --

password: ••••••••

Without any check

1

Using static analysis

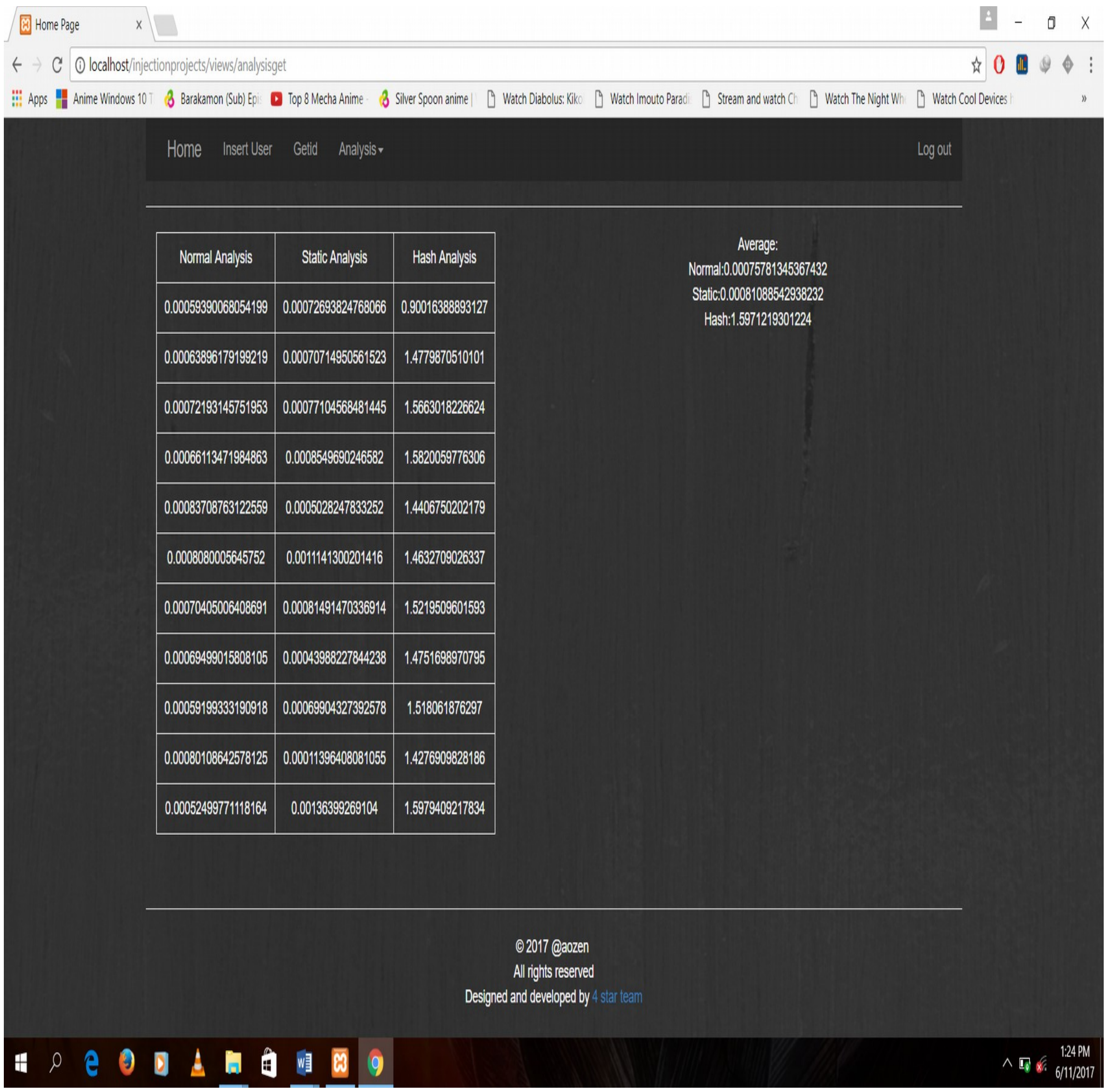
On username:quote Detected
Bad input detected
out of the range of good inputs
On password:

Using hash function

Incorrect username or password

© 2017 @aozen
All rights reserved
Designed and developed by 4 star team

Figure 5.4(d): Injection try



Normal Analysis	Static Analysis	Hash Analysis
0.00059390068054199	0.00072693824768066	0.90016388893127
0.00063896179199219	0.00070714950561523	1.4779870510101
0.00072193145751953	0.00077104568481445	1.5663018226624
0.00066113471984863	0.0008549690246582	1.5820059776306
0.00083708763122559	0.0005028247833252	1.4406750202179
0.0008080005645752	0.0011141300201416	1.4632709026337
0.00070405006408691	0.00081491470336914	1.5219509601593
0.00069499015808105	0.00043988227844238	1.4751698970795
0.00059199333190918	0.00069904327392578	1.518061876297
0.00080108642578125	0.00011396408081055	1.4276909828186
0.00052499771118164	0.00136399269104	1.5979409217834

Average:
Normal:0.00075781345367432
Static:0.00081088542938232
Hash:1.5971219301224

© 2017 @aozen
All rights reserved
Designed and developed by 4 star team

Figure 5.4(e): Time analysis

CHAPTER VI: CONCLUSION AND FUTURE ENHANCEMENT

6.1 Conclusion

The main intention of our system is to give proper analysis of the algorithms we have implemented for the detection of SQL injections. The main focus of our system is to block the SQL injection attacks for securing the database. The results after comparing against the attacks using the algorithms were found to be satisfactory.

After the development of this project we have learned about the static and hash algorithms built for the purpose of defending against the SQL injection attacks and compare the result of each algorithm to give a proper analysis.

6.2 Limitation and future enhancement

6.2.1 Limitation

1. Our system mainly focuses on SQL injection attacks only.

2. The hash algorithm used cannot distinguish the type of SQL injection that has been injected.

3. The static algorithm used cannot defend against the new type of injection attacks.

3. Other vulnerabilities like Cross Site Scripting, Broken Session Control, etc. is not handled by our system.

6.2.2 Future Enhancement

1. We will implement more hash functions for more improved security.

2. The static algorithm used will be updated on a regular basis.

3. We will expand our horizon and make our program potent to other attacks as well.

REFERENCES

[1] K. Bagchi and G. Udo, “An analysis of the growth of computer and internet security breaches,” Communications of the Association for Information Systems, vol. 12, no. 1, p. 46, 2003.

[2] *Sean Michael Kerner (November 25, 2013).* "How Was SQL Injection Discovered? The researcher once known as Rain Forrest Puppy explains how he discovered the first SQL injection more than 15 years ago."

[3] Allam Appa Rao, P.Srinivas, B. Chakravarthy, K.Marx, and P. Kiran “A Java Based Network Intrusion Detection System (IDS)"

APPENDIX

Source Code:

A. Code for User Interface

//login form

```
<form method="POST" action="login_process.php" class="form-horizontal">
        <div class="form-group">
                <label class="col-sm-2 control-label">UserName:</label>
                <div class=" col-sm-4"><input type="text" name="uname" class="form-control" required></div>
        </div>
        <div class="form-group">
                <label class="col-sm-2 control-label">Password:</label>
                <div class=" col-sm-4"><input type="password" name="upass" class="form-control col-sm-8" required></div>
        </div>
                <div class="col-sm-6"><input type="submit" class = "btn btn-default" name="submit" value="login"></div>
        </form>
```

//login process

```
if(!isset($_SESSION['user']))

{

        if(isset($_POST['submit']))

        {



                $user = htmlspecialchars($_POST['uname']);

                $pass = htmlspecialchars($_POST['upass']);



                $check = $normaldata->get_data($user, $pass);



                if ($check>=1)

                {

                        $_SESSION['user'] = $user;

                        header('location:../home.crypt');

                }

                $_SESSION['msg'] = " UserName or password Error";

        }

                header('location:userlogin');

}

else

        header('location:../home.crypt');
```

```
//log out process

        if(isset($_SESSION['user']))

        {

                unset($_SESSION['user']);

                session_destroy();

                session_start();

                $_SESSION['msg'] = "Logged Out";

        }

        header('location:userlogin');
```

B. Code for static method

```
// check username and password

        public function get_data($username, $password)

        {

                $query = "Select * from userid where username = '".$username."' and
password = '".$password."'";// select query

                        if($this->conn->query($query))

                        {

                                $result = $this->conn->query($query);

                                $row = mysqli_fetch_assoc($result);
```

```
                    if($result->num_rows>0)// check for rows to return
                        return $row['id'];
                    else
                        return -1;//if no rows are found
                }
                else
                    return 0;
        }



        public function getstatic_data($username, $password)
        {
            $detect_on_username = $this->detect($username);
            $detect_on_password = $this->detect($password);
            if($detect_on_username==null&&$detect_on_password==null)
            {
                return $this->get_data($username, $password);
            }
            else
                return "On username:".$detect_on_username."<br>On password:".
    $detect_on_password;
```

```
		}

	//detect inputs
		public function detect($query)
		{
			$return = null;
			if($this->detect_quotes($query))//detect quotes
				$return = "quote Detected";
			if($this->detect_bad_input($query))//detect sql query
				$return = $return."<br> Bad input detected ";
			if($this->detect_logical($query))
				$return = $return."<br> Logical operator detected ";
			if($this->detect_good_inputs($query))
				$return = $return."<br> out of the range of good inputs ";
			return $return;
		}

	//check for logical operators
		public function detect_logical($query)
		{
			$return = 0;
```

```
                $detect = array("or", "and", "||", "&&");

                $split = explode(" ", $query);

                $size = sizeof($split);

                echo strcmp(strtolower($split[0]),$detect[0]);

                for( $i=0;$i<$size;$i++)

                {

                        for($j =0;$j<2;$j++)

                        {

                                if(strcmp(strtolower($split[$i]),$detect[$j])==0)//compare
for logical operators

                                {

                                        $return = 1;

                                        break;

                                }

                        }

                }

                return $return;

        }


//check for quotes

        public function detect_quotes($query)
        {
                if (preg_match('/"/',$query)||preg_match("/'/",$query))//check for any
quotes
                        return 1;
                else
                        return 0;
        }


//check for the sql syntaxs
        public function detect_bad_input($query)
        {
                $return = 0;
                $detect = array("union", "select", "update", "insert", "delete", "--", "drop",
"=");//bad inputs for query
                $split = explode(" ", $query);//split the query


                $querysize = sizeof($split);
                $detectsize = sizeof($detect);


                for( $i=0;$i<$querysize;$i++)
                {
                        for($j =0;$j<$detectsize;$j++)
                {
                    if(strcmp(strtolower($split[$i]),$detect[$j])==0)//compare
from the detection query
                    {
                        $return = 1;
                        break;
                    }
                }
            }
            return $return;
        }

//check for only good inputs
        public function detect_good_inputs($query)
        {
            $return = 1;
            echo "ps ".preg_match('/[^a-zA-Z0-9]/', $query);
            if(!preg_match('/[^a-zA-Z0-9]/', $query))
                $return = 0;
            return $return;
        }
```

```
		}
```

C. Code for Hash Data

```
// check username and password
		public function getfrom_hashdata($username, $password)
		{
			$query = "Select * from hashuserid";// select query
			if($this->conn->query($query))
			{
				$result = $this->conn->query($query);
				if($result->num_rows>0)// check for rows to return
				{
					while($row = mysqli_fetch_assoc($result))
					{
					if(password_verify($username,
$row['hash_username'])&&password_verify($password, $row['hash_password']))
						return $row['id'];
					}
				}
				else
					return -1;//if no rows are found
		}
		else
			return 0;
	}

	public function insert_hash_data($username, $password)
	{
		$uname = password_hash($username, PASSWORD_DEFAULT);
		$upass = password_hash($password, PASSWORD_DEFAULT);
		$query = "Insert into hashuserid (hash_username, hash_password) values
('".$uname."', '".$upass."')";
		if($this->conn->query($query))
		{
			return "Hash Insert Successful";
		}
		else
			return "Error on Hash insertion";
	}
}
```

Snapshot

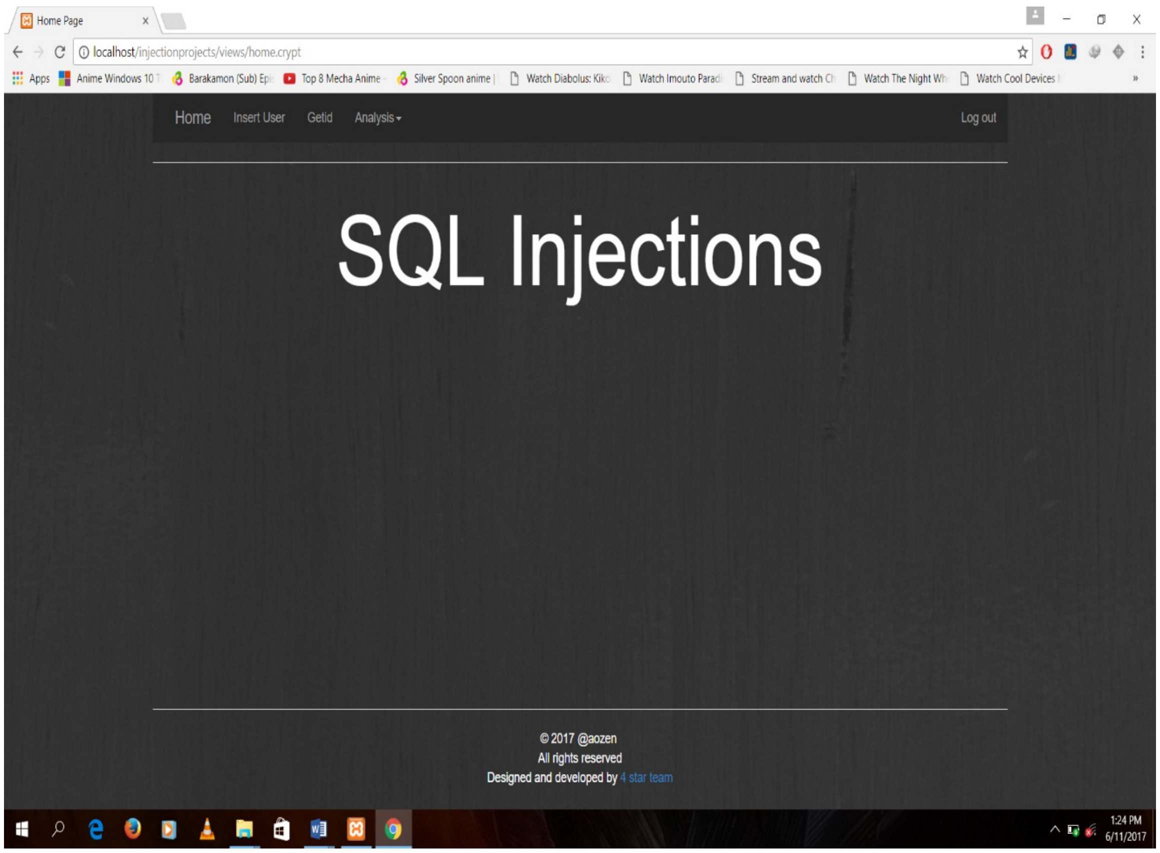


Figure: User interface

Time line of project

Task Description	Weeks												
	March				April				May				June
	1	2	3	4	1	2	3	4	1	2	3	4	1
Research and information gathering	🟧	🟧	🟧	🟧	🟧	🟧	🟧						
Project plan		🟧	🟧	🟧	🟧	🟧							
Coding					🟧	🟧	🟧	🟧	🟧	🟧	🟧		
Testing					🟧	🟧	🟧	🟧	🟧	🟧	🟧		
System Implementation										🟧	🟧	🟧	
Documentation	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧

Graphs

Retrieving data From Database

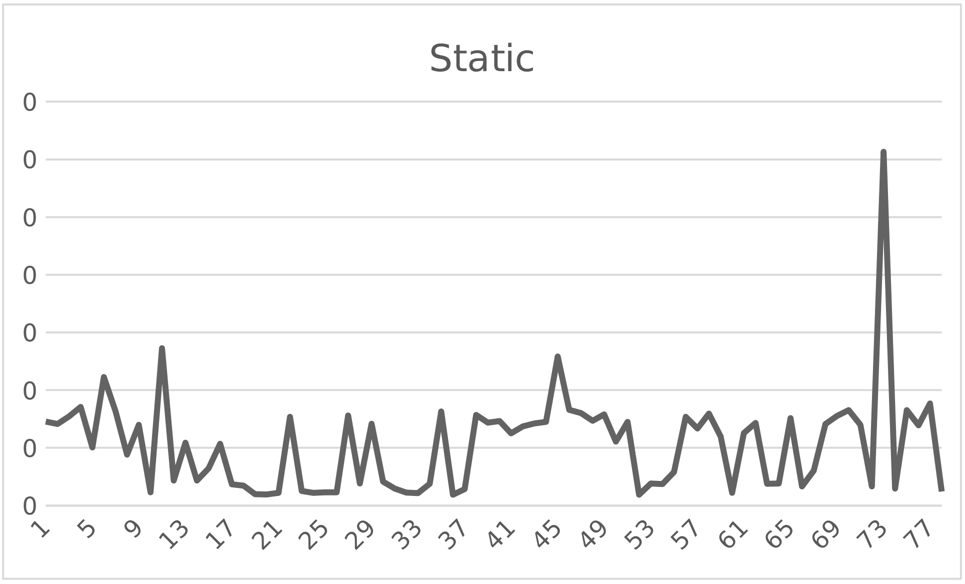


Figure: Time Analysis of static Algorithm

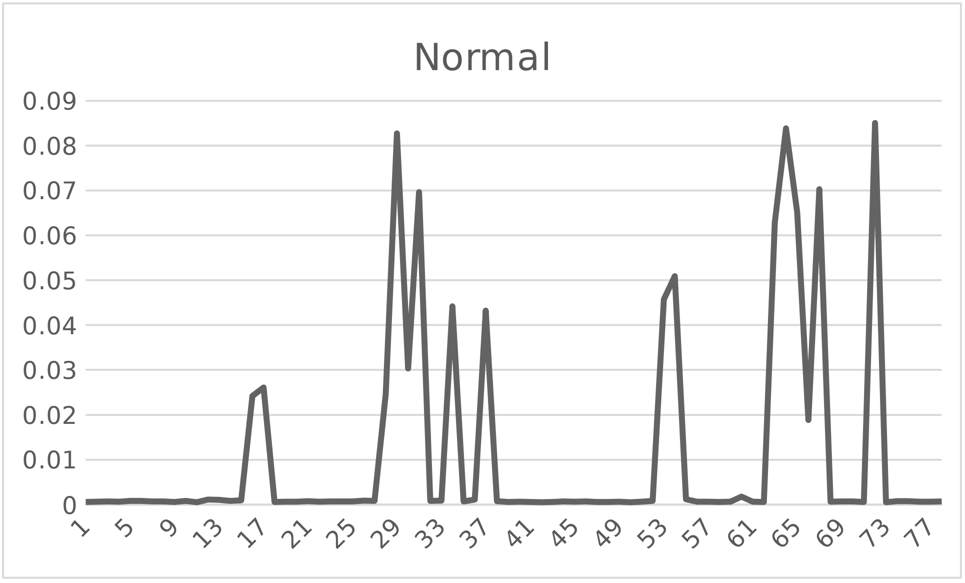


Figure: Time Analysis of Normal retrieve of data

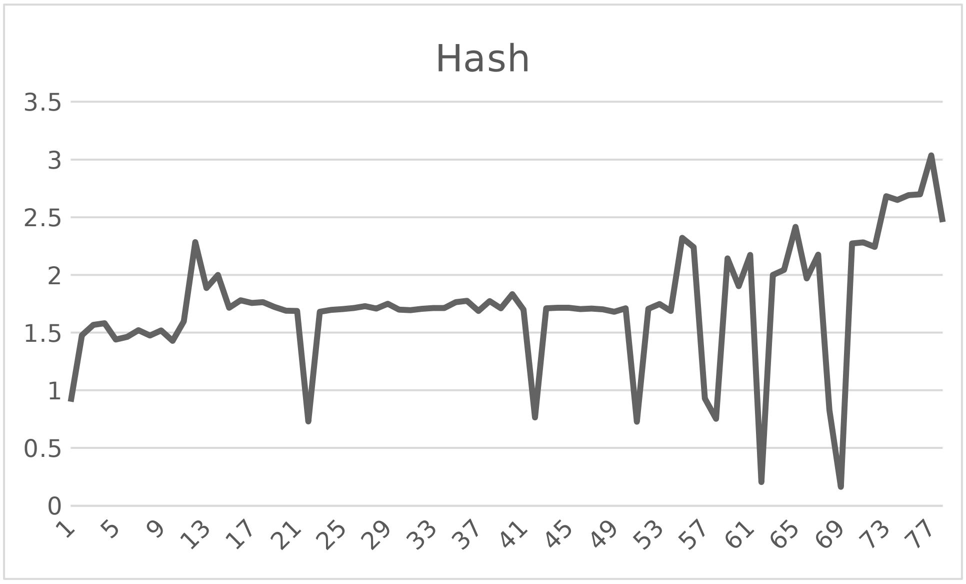


Figure: Time analysis of hash algorithm

Inserting into Database

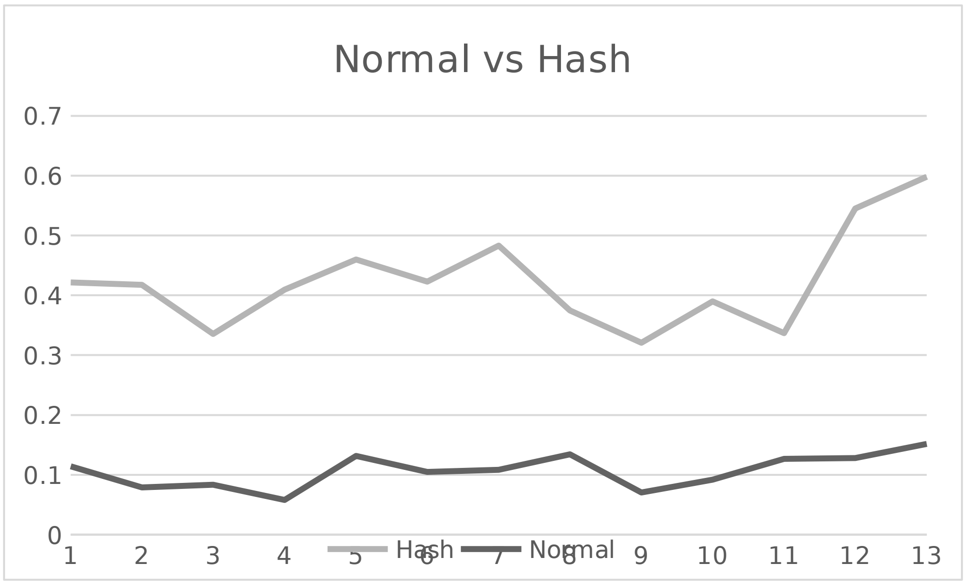


Figure: Time analysis of insertion