FORESTALLMENT OF SQL INJECTION INE-COMMERCE

J.Jagadeswara Reddy[1] , Assistant professor in Dept of Computer Science Engineering.

Dr Pandurangan Ravi[2*] , Professor in Dept of Computer Science Engineering.

[1,2*]Sai Rajeswari Institute of Technology, Proddatur, Kadapa, Andhra Pradesh, 516362, India

ABSTRACT

The ideal of this design is to help SQL injection while firing queries to database and to make the database secured. This system is online so need of perpetration. It can be penetrated through internet from anywhere. The system uses SQL injection medium to keep the data safe and secured. The stressed part then's encryption of card data using AES(Advanced Encryption Standard) fashion SQL injection is a type which the bushwhacker adds SQL law to a web from input box to gain access or make change to data. The system uses SQL injection medium to keep data safe and secured. Re-searches have proposed different tools to descry and help SQL Vulnerability. The design will be penetrated in the web cyber surfer through Azure link.

Keywords: Vulnerability, Detection, Prevention, Intruder, SQL injection attacks

OBJECTIVE

This project aims at developing an security system that will helps to an authorized person with a easy manner. The unauthorized persons cannot access the data. To enhance understanding of SQL injection, it is better to have good understanding of the kinds of communications that take place during a typical session between a user and a web based

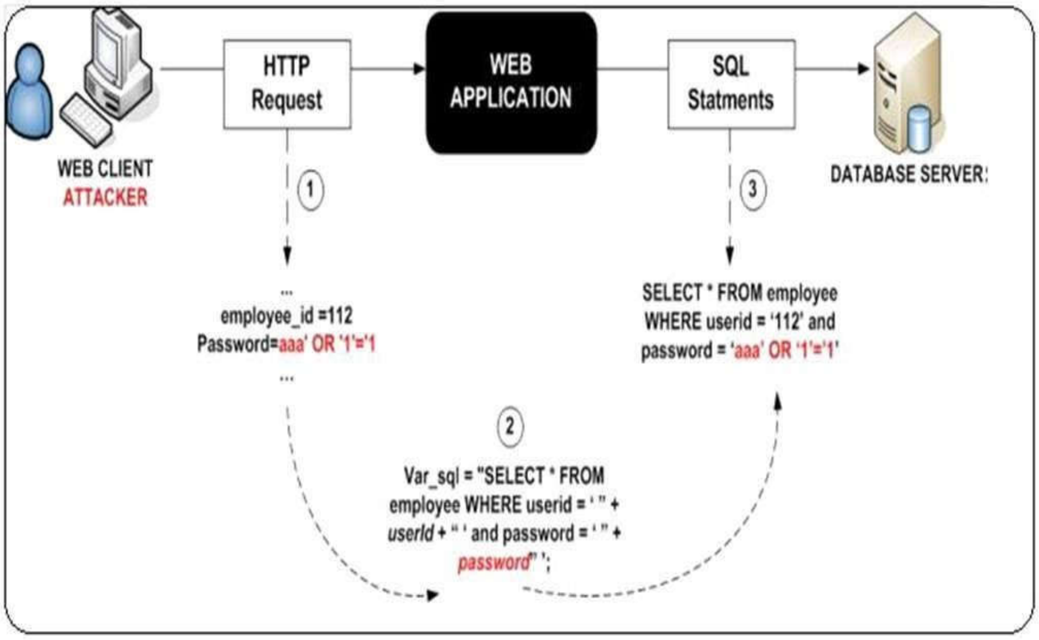
METHODOLOGY

SQL injection is widely used hacking technique, in which the intruder adds SQL statements using a web application as input fields to access to the secret users resources and due to lack of input Validation in web applications causes intruders to be successful in hacking. With above said technique, we can assume that a Web application receives “http//” request from a user client as Input and generates a SQL statement as output for the back-End database server. For example an administrator will be authenticated after providing input as -Typing: employee id - 0112 and password =admin, configure .That describes a login by a suspicious user exploiting SQL Injection vulnerability. Usually, it is structured in three phases,

(1) An Intruder sends the malicious “http//” request to the Web application,

(2). Generates the statement,

(3). Dedicatedly Deposited the SQL statement to the back end database.


WEB CLIENT
ATTACKER
HTTP Request
WEB APPLICATION
SQL Statments
DATABASE SERVER
1
...
employee_id =112
Password=aaa' OR '1'='1
...
3
SELECT * FROM employee WHERE userid = '112' and password = 'aaa' OR '1'='1'
2
Var_sql = "SELECT * FROM employee WHERE userid = ' " + userId + " ' and password = ' " + password" ';

Methodology

Advantages:

1. Secured transactions while doing card payments.
2. Less risk of data getting hacked.
3. The system is very secure and robust in nature.
4. SQL injection prevention mechanism is used.
5. Data is kept secured on cloud server which avoids unauthorized access.

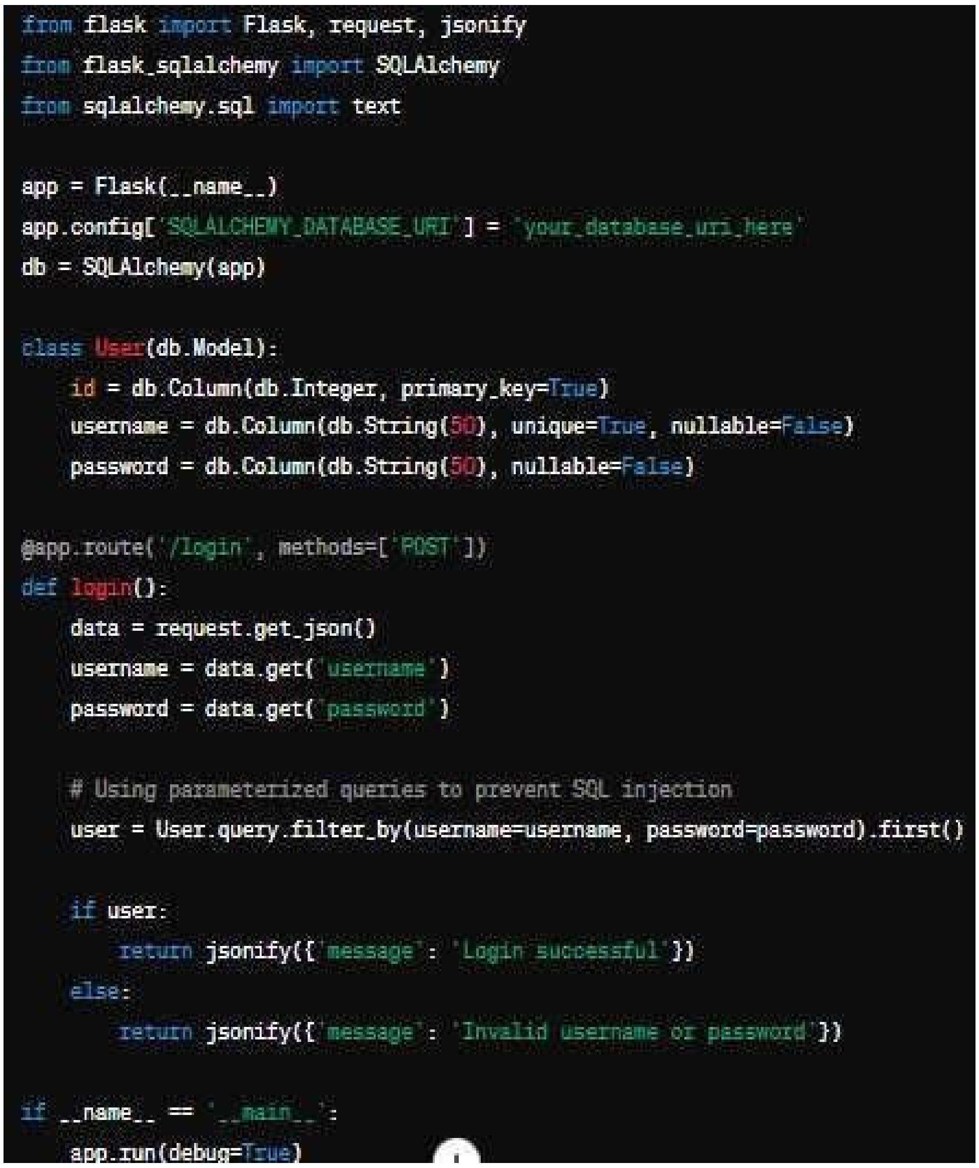
Disadvantages:

1. Does not keep track of stock/order.
2. System may provide inaccurate results if data entered incorrectly.
3. Needs active internet connection to connect with cloud server.
4. Server issues

SQL SOURCE CODE

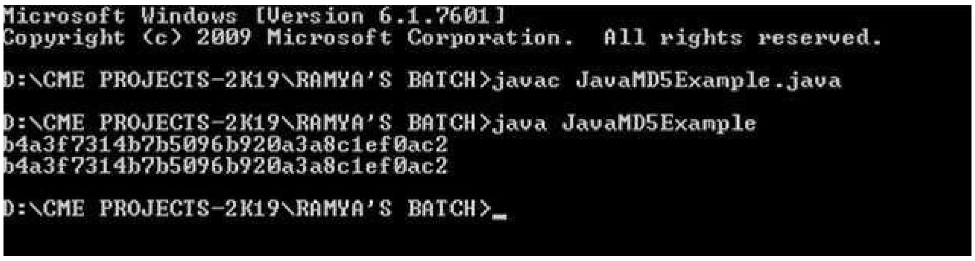
import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;



```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.sql import text

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'your_database_uri_here'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(50), nullable=False)

@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    # Using parameterized queries to prevent SQL injection
    user = User.query.filter_by(username=username, password=password).first()

    if user:
        return jsonify({'message': 'Login successful'})
    else:
        return jsonify({'message': 'Invalid username or password'})

if __name__ == '__main__':
    app.run(debug=True)
```

Output:



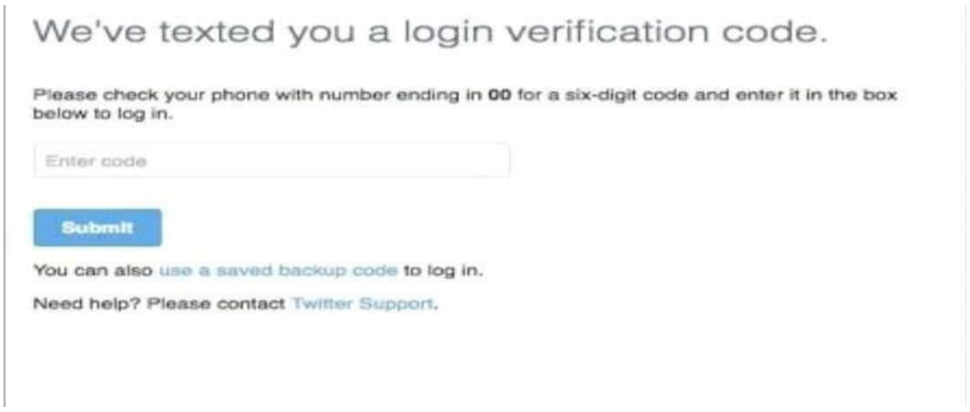
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

D:\CME PROJECTS-2K19\RAMYA'S BATCH>javac JavaMD5Example.java

D:\CME PROJECTS-2K19\RAMYA'S BATCH>java JavaMD5Example
b4a3f7314b7b5096b920a3a8c1ef0ac2
b4a3f7314b7b5096b920a3a8c1ef0ac2

D:\CME PROJECTS-2K19\RAMYA'S BATCH>_
```

RESULT:

Current status:

LEVEL : 0
User ID
Password
OK
Cancel

Verification code:


We've texted you a login verification code.

Please check your phone with number ending in 00 for a six-digit code and enter it in the box below to log in.

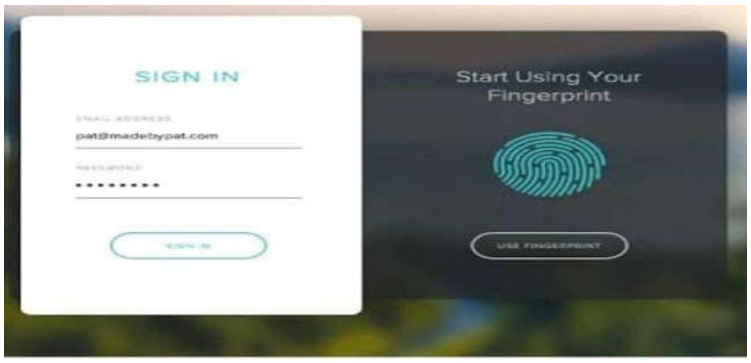
Enter code

Submit

You can also use a saved backup code to log in.

Need help? Please contact Twitter Support.

Fingerprint:



FUTURE ENHANCEMENT:

For people, likewise, it can be enforced to be used for SQL query profiling, SQL query table and discovery program modularization. unborn work is demanded on this exploration for not only SQL Injection attacks but also other web operation attacks similar as XSS. These discovery tools can be used to descry SQL Injection attacks. These ways can also give as defense mechanisms for furnishing security against SQLIAs. In addition, further exploration is demanded to ameliorate analysis fashion for furnishing better discovery and forestallment against strong

SQLIAs. In our unborn work we separate ways which have been enforced as tool, also compare effectiveness, effectiveness, stability, inflexibility and performance of tools to show the strength and weakness of the tool.

REFERENCES

[1] K. Rajeswari, C. Amsaveni" SQL injection Attack Prevention Using 448 Blowfish Encryption Standard", International Journal of Computer Science Trends and technology, Vol. 4,325-335, August 2016.

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

[3] Avireddy. S, Perumal.V, Gowraj.N, Kannan R.S, Thinakaran.P, Ganapthi .S, Gunasekaran J.R, Prabhu.S, "Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL injection", In: 11th IEEE International conference on Trust, Security and Privacy in computing and communications, 1327-1333, 2012

[4] The Open Web Application Security Project, "OWASP TOP 10 Projects" http://www.owasp.org/ .

[5] PHP, magic quotes, http://www.php.net/magic_quotes/ .

[6] Apache Struts project, Struts. http://struts.apache.org/ .

[7] C. Gould, Z. Su, P. Devanbu, "JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications", In Proceedings of the 26th International Conference on Software Engineering (ICSE), pp. 697-698, 2004.

[8] William. G. Halfond, Alessandro. Orso, "Using Positive Tainting and Syntax Aware Evaluation to Counter SQL Injection Attacks", 14th ACM SIGSOFT international symposium on Foundations of software engineering 2006, ACM