SQL injection:SQL Injection is one of the top 10 web application vulnerabilities.SQL Injection means injecting/inserting SQL code in a query via user-inputted data. It can occur in any applications using relational databases like Oracle, MySQL, PostgreSQL and SQL Server.To perform SQL Injection, a malicious user first tries to find a place in the application where he can embed SQL code along with data. It can be the login page of any web application or any other place. So when data embedded with SQL code is received by the application, SQL code will be executed along with the application query.

select \* from app.employee where empid='101'.

Works:Suppose we have a database table named tbluser which stores data of application users. The userId is the primary column of the table. We have functionality in the application, which lets you get information via userId. The value of userId is received from the user request.

Let’s have a look at the below example code.

String userId = {get data from end user};

String sqlQuery = "select \* from tbluser where userId = " + userId;

1.valid user input:When the above query is executed with valid data i.e. userId value 132, it will look like below.

Input Data: 132

Executed Query: select \* from tbluser where userId=132

Result: Query will return data of user having userId 132. No SQL Injection is happening in this case.

2. Hacker User Input:A hacker can alter user requests using tools like Postman, cURL, etc. to send SQL code as data and this way bypassing any UI side validations.

Input Data: 2 or 1=1

Executed Query: select \* from tbluser where userId=2 or 1=1

Result: Now the above query is having two conditions with SQL OR expression.

userId=2: This part will match table rows having userId value as ‘2’.

1=1: This part will be always evaluate as true. So Query will return all the rows of the table.

Preventing SQL Injection in Java Code:

The simplest solution is to use PreparedStatement instead of Statement to execute the query.Instead of concatenating username and password into the query, we provide them to query via PreparedStatement’s setter methods.

Example:public static void main(String[] args){

try{

class.hasName("org.apache.derby.jdbc.ClintDriver");

Connection connection=DriverManager.getConnection("jdbc:derby://localhost:1527/sample;create=true","user"."user");

PreparedStatement ps = conn.prepareStatement("select \* from app.employee where empid=?");

ps.setString(1,empid);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

System.out.println(rs.getString(1) + "--" + rs.getString(2));

}

}

}

Statements::Once a connection is obtained we can interact with the database. The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

1.Statement 2.PreparedStatement 3.CallableStatement

1.Statement:Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement( ) method.Once we created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

Boolean execute (String SQL): Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.

Int executeUpdate (String SQL) − Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.

ResultSet executeQuery (String SQL) − Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

Closing Statement Object:

Just as you close a Connection object to save database resources, for the same reason you should also close the Statement object.A simple call to the close() method will do the job. If you close the Connection object first, it will close the Statement object as well. However, you should always explicitly close the Statement object to ensure proper cleanup.

Example::public static void main(String[] args) {

try {

Class.forName("org.apache.derby.jdbc.ClientDriver");

Connection connection = DriverManager.getConnection("jdbc:derby://localhost:1527/sample;create=true","user","user");

Statement st = connection.createStatement();

ResultSet rs = st.executeQuery("select \* from app.employee");

st.execute();

System.out.println("Employee Details Saved Successfullyy..");

while(rs.next()){

System.out.println(rs.getString(1));

System.out.println(rs.getString(2));

}

connection.close();

} catch (ClassNotFoundException |SQLException se) {

se.printStackTrace();

}

}

}

2.PreparedStatement:The PreparedStatement interface extends the Statement interface, which gives the added functionality with a couple of advantages over a generic Statement object.This statement gives the flexibility of supplying arguments dynamically.All parameters in JDBC are represented by the ? symbol, which is known as the place holders.Wee have to supply values for every parameter before executing the SQL statement.If we forget to supply the values, we will receive an SQLException.All of the Statement object's methods for interacting with the database (a) execute(), (b) executeQuery(), and (c) executeUpdate() also work with the PreparedStatement object. However, the methods are modified to use SQL statements that can input the parameters.SQL injection is handled by preparedStatement. If we want to close the Connection object first, it will close the PreparedStatement object as well. However, we should always explicitly close the PreparedStatement object to ensure proper cleanup.

Example:public static void main(String[] args) {

try {

Class.forName("org.apache.derby.jdbc.ClientDriver");

Connection connection = DriverManager.getConnection("jdbc:derby://localhost:1527/sample;create=true","user","user");

PreparedStatement st = connection.prepareStatement("insert into app.employee values(?,?)");

Scanner sc = new Scanner(System.in);

System.out.println("Enter EmpId: ");

String empId = sc.next();

System.out.println("Enter EmpName: ");

String empNm = sc.next();

st.setString(1, empId);

st.setString(2, empNm);

st.execute();

System.out.println("Employee Details Saved Successfullyy..");

connection.close();

} catch (ClassNotFoundException |SQLException se) {

se.printStackTrace();

}

}

3.CallableStatement:Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute a call to a database stored procedure.Using the CallableStatement objects is much like using the PreparedStatement objects. We must bind values to all the parameters before executing the statement, or we will receive an SQLException.

Example:public class token {

public static void main(String[] args) {

Class.forName("org.apache.derby.jdbc.ClientDriver");

Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/MyTestDb;create=true", "derby",

"derby");

System.out.println("Enter Employee Id: ");

Scanner sc = new Scanner(System.in);

String empid = sc.nextLine();

PreparedStatement ps = conn.prepareStatement("select \* from app.employee where empid=?");

ps.setString(1,empid);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

System.out.println(rs.getString(1) + "--" + rs.getString(2));

}

CallableStatement cs = conn.prepareCall("exec func myFunction(?,?)");

cs.setInt(1,10);

cs.setInt(2,20);

cs.executeQuery();

}

}

Inner Class Types:The inner class is a class that is declared inside the class or interface.We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.users need to program a class in such a way so that no other class can access it. Therefore, it would be better if you include it within other classes.If all the class objects are a part of the outer object then it is easier to nest that class inside the outer class. That way all the outer class can access all the objects of the inner class.

There are 4 types of inner classes:

1.Normal inner class 2.Static inner class 3.Anonymous inner class 4.Method inner class.

Normal innner class::A class created within class and outside method.A non-static class that is created inside a class but outside a method is called member inner class. It is also known as a regular inner class. It can be declared with access modifiers like public, default, private, and protected.

public class Mycar {

public static void main(String[] args){

Car car=new Car();

Car.Engine engine=car.new Engine();

engine.myMethod();

Car.Brake brake=new Car.Brake();

brake.brMethod();

}

}

class Car{

class Engine{

public void myMethod(){

System.out.println("i am engine");

}

}

Static inner class::A static class is a class that is created inside a class, is called a static nested class in Java. It cannot access non-static data members and methods. It can be accessed by outer class name.It can access static data members of the outer class, including private.The static nested class cannot access non-static (instance) data members.A static class is a class that is created inside a class, is called a static nested class in Java. It cannot access non-static data members and methods.In this example,we need to create the instance of static nested class because it has instance method. But we don't need to create the object of the Outer class because the nested class is static and static properties, methods, or classes can be accessed without an object.

Example::public class Mycar {

public static void main(String[] args){

Car car=new Car();

Car.Engine engine=car.new Engine();

engine.myMethod();

Car.Brake brake=new Car.Brake();

brake.brMethod();

}

}

class Car{

class Engine{

public void myMethod(){

System.out.println("i am engine");

}

}

static class Brake{

public void brMethod(){

System.out.println("i am brake");

}

}

}

Anonymous inner class::Java anonymous inner class is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain "extras" such as overloading methods of a class or interface, without having to actually subclass a class.In simple words, a class that has no name is known as an anonymous inner class in Java. It should be used if you have to override a method of class or interface. Java Anonymous inner class can be created in two ways:

\*\*Class (may be abstract or concrete).

\*\*Interface

Example:public class MyInter {

public static void main(String[] args) throws Exception {

Myanony an=new Myanony(){

public void myMethod(){

System.out.println("i am anounymous");

}

};

an.myMethod();

}

}

interface Myanony{

public void myMethod();

}

Method inner class::In Java, we can write a class within a method and this will be a local type. Like local variables, the scope of the inner class is restricted within the method.A method-local inner class can be instantiated only within the method where the inner class is defined. The following program shows how to use a method-local inner class.

Example::package method;

public class MethodinnerClass {

public static void main(String args[]) {

MethodinnerClass outer = new MethodinnerClass();

outer.my\_Method();

}

void my\_Method() {

int num =5;

class MethodInner {

public void print() {

System.out.println("This is method inner class "+num);

}

}

MethodInner inner = new MethodInner();

inner.print();

}

}

Realationships in Java: There are three most common relationships among classes in Java that are as follows:

a. Uses-A ------- Dependence

b. Has-A ------- Association

c. Is-A ------- Inheritance

Uses-A:When we create an object of a class inside a method of another class, this relationship is called dependence relationship in Java, or simply Uses-A relationship.If several classes of an application program depend on each other, then we say that the coupling between classes is high.

Example:class Car{

}

class maruthi{

void disp(){

Car obj=new car();

}

}

Has-A:Association is another fundamental relationship between classes that is informally known as “Has-A” relationship.When an object of one class is created as data member inside another class, it is called association relationship in java or simply Has-A relationsh

Example:class Car{

}

class Maruth{

Car obj=new Car();

}

Is-A:Inheritance represents Is-a relationship in Java. It establishes a relationship between a more general class and a more specialized class (known as subclass).In other words, Is-A relationship defines the relationship between two classes in which one class extends another class.

Example:class Car{

}

class Maruthi extends Car{

}