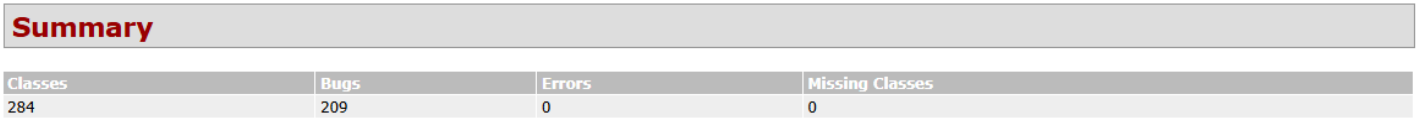
Brendan Helms

700695073

Webgoat + SpotBug Project

Abstract: In this project, I will analyze the vulnerable Webgoat program. Webgoat is maintained by OWASP and is meant to be a teaching tool for those interested in rectifying vulnerabilities. The tool used for the analysis of Webgoat is the static analysis tool SpotBugs. This tool focuses on predetermined patterns to identify common vulnerabilities within a program. Spotbugs can also be used as a standalone tool but in this project it is used through maven. My initial spotbugs report determined that there were a total of 209 bugs found within Webgoat. In the following steps, I will identify 5 bugs of the Security category, rectify the code, and run a new spotbugs scan to ensure that the bug has been removed.

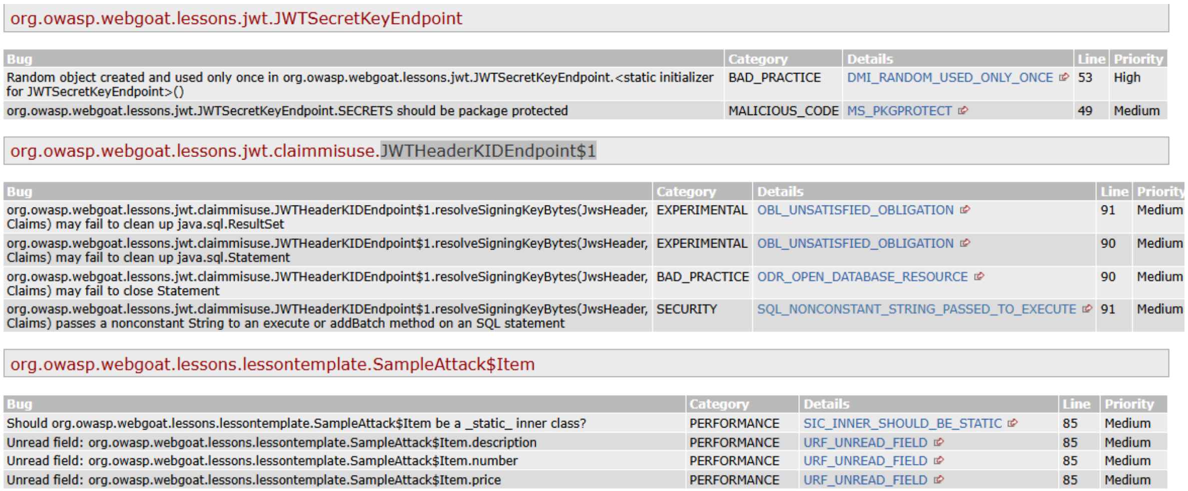


Summary

Classes	Bugs	Errors	Missing Classes
284	209	0	0

1. Bug #1

a. Initial Bug Report



org.owasp.webgoat.lessons.jwt.JWTSecretKeyEndpoint

Bug	Category	Details	Line	Priority
Random object created and used only once in org.owasp.webgoat.lessons.jwt.JWTSecretKeyEndpoint.<static initializer for JWTSecretKeyEndpoint>()	BAD_PRACTICE	DMI_RANDOM_USED_ONLY_ONCE	53	High
org.owasp.webgoat.lessons.jwt.JWTSecretKeyEndpoint.SECRETS should be package protected	MALICIOUS_CODE	MS_PKGPROTECT	49	Medium

org.owasp.webgoat.lessons.jwt.claimmisuse.JWTHeaderKIDEndpoint$1

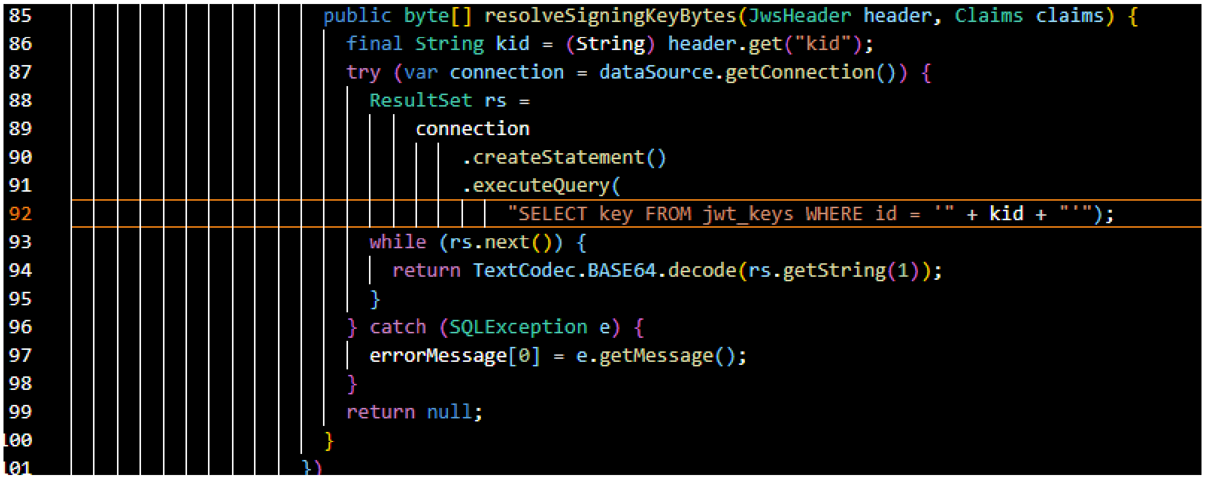
Bug	Category	Details	Line	Priority
org.owasp.webgoat.lessons.jwt.claimmisuse.JWTHeaderKIDEndpoint$1.resolveSigningKeyBytes(JwsHeader, Claims) may fail to clean up java.sql.ResultSet	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	91	Medium
org.owasp.webgoat.lessons.jwt.claimmisuse.JWTHeaderKIDEndpoint$1.resolveSigningKeyBytes(JwsHeader, Claims) may fail to clean up java.sql.Statement	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	90	Medium
org.owasp.webgoat.lessons.jwt.claimmisuse.JWTHeaderKIDEndpoint$1.resolveSigningKeyBytes(JwsHeader, Claims) may fail to close Statement	BAD_PRACTICE	ODR_OPEN_DATABASE_RESOURCE	90	Medium
org.owasp.webgoat.lessons.jwt.claimmisuse.JWTHeaderKIDEndpoint$1.resolveSigningKeyBytes(JwsHeader, Claims) passes a nonconstant String to an execute or addBatch method on an SQL statement	SECURITY	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	91	Medium

org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item

Bug	Category	Details	Line	Priority
Should org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item be a _static_ inner class?	PERFORMANCE	SIC_INNER_SHOULD_BE_STATIC	85	Medium
Unread field: org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item.description	PERFORMANCE	URF_UNREAD_FIELD	85	Medium
Unread field: org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item.number	PERFORMANCE	URF_UNREAD_FIELD	85	Medium
Unread field: org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item.price	PERFORMANCE	URF_UNREAD_FIELD	85	Medium

b. **Bug Description:** This bug indicates the possibility of an SQL injection within line 91 of the code. This is due to a dynamic string being directly concatenated to the SQL query statement. A dynamic string is created when a string is concatenated with variables. This is a security risk because an attacker could manipulate the variable directly in the SQL query. This could then compromise sensitive data and confidentiality.

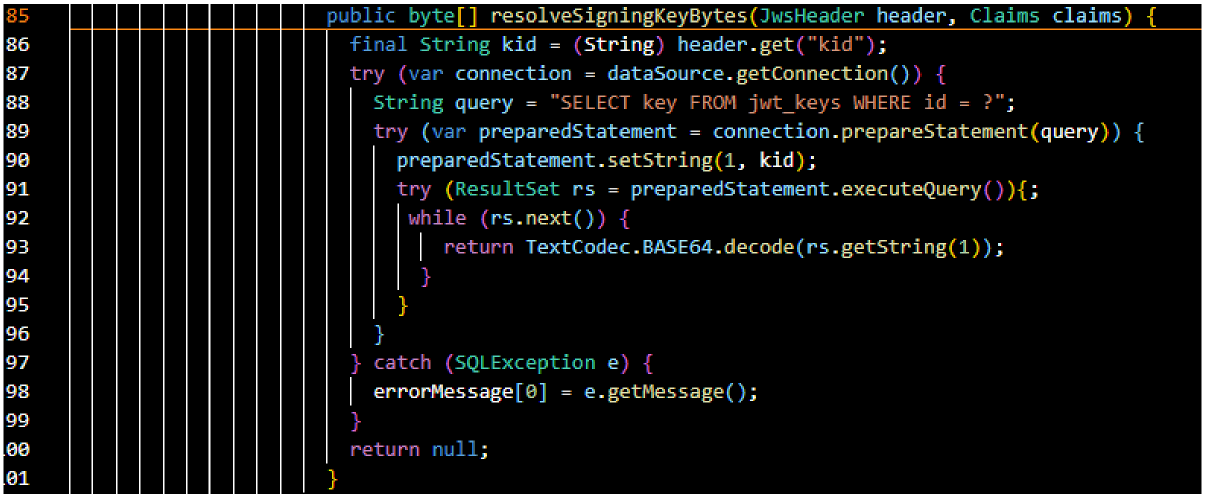
The dynamic string in this case occurs on lines 91-92. The kid variable being concatenated directly into the query increases the risks of SQL injection.



```
85  public byte[] resolveSigningKeyBytes(JwsHeader header, Claims claims) {
86    final String kid = (String) header.get("kid");
87    try (var connection = dataSource.getConnection()) {
88      ResultSet rs =
89          connection
90              .createStatement()
91              .executeQuery(
92                  "SELECT key FROM jwt_keys WHERE id = '" + kid + "'");
93      while (rs.next()) {
94        return TextCodec.BASE64.decode(rs.getString(1));
95      }
96    } catch (SQLException e) {
97      errorMessage[0] = e.getMessage();
98    }
99    return null;
100 }
101 })
```

c. **Edited Code:**

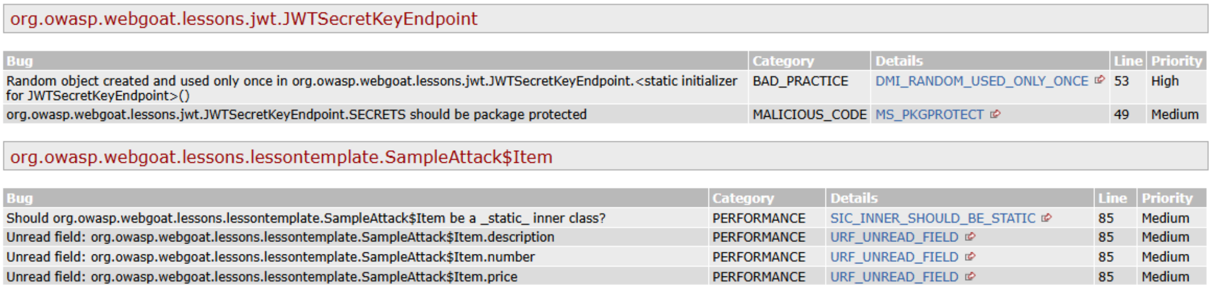
To fix the problem I removed the dynamic statement and replaced it with a prepared statement. The '?' acts as a placeholder in the query and the database will know that future values used are data and not executable queries (which could be malicious).



```
public byte[] resolveSigningKeyBytes(JwsHeader header, Claims claims) {
  final String kid = (String) header.get("kid");
  try (var connection = dataSource.getConnection()) {
    String query = "SELECT key FROM jwt_keys WHERE id = ?";
    try (var preparedStatement = connection.prepareStatement(query)) {
      preparedStatement.setString(1, kid);
      try (ResultSet rs = preparedStatement.executeQuery()){;
       while (rs.next()) {
          return TextCodec.BASE64.decode(rs.getString(1));
        }
      }
    }
  } catch (SQLException e) {
    errorMessage[0] = e.getMessage();
  }
  return null;
}
```

d. Updated Bug Report

The ‘org.owasp.webgoat.lessons.jwt.claimmisuse.JWTHeaderKIDEndpoint’ section no longer exists as seen in the screenshot of the updated spotbugs report.



org.owasp.webgoat.lessons.jwt.JWTSecretKeyEndpoint

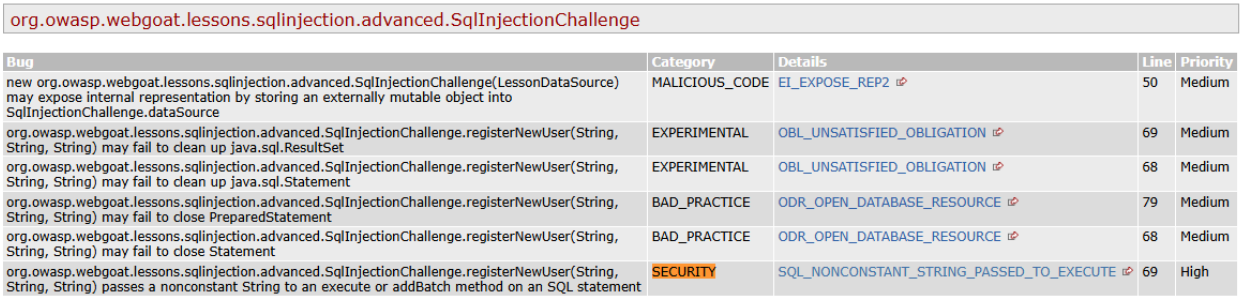
Bug	Category	Details	Line	Priority
Random object created and used only once in org.owasp.webgoat.lessons.jwt.JWTSecretKeyEndpoint.<static initializer for JWTSecretKeyEndpoint>()	BAD_PRACTICE	DMI_RANDOM_USED_ONLY_ONCE	53	High
org.owasp.webgoat.lessons.jwt.JWTSecretKeyEndpoint.SECRETS should be package protected	MALICIOUS_CODE	MS_PKGPROTECT	49	Medium

org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item

Bug	Category	Details	Line	Priority
Should org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item be a _static_ inner class?	PERFORMANCE	SIC_INNER_SHOULD_BE_STATIC	85	Medium
Unread field: org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item.description	PERFORMANCE	URF_UNREAD_FIELD	85	Medium
Unread field: org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item.number	PERFORMANCE	URF_UNREAD_FIELD	85	Medium
Unread field: org.owasp.webgoat.lessons.lessontemplate.SampleAttack$Item.price	PERFORMANCE	URF_UNREAD_FIELD	85	Medium

2. Bug #2

a. Initial Bug Report

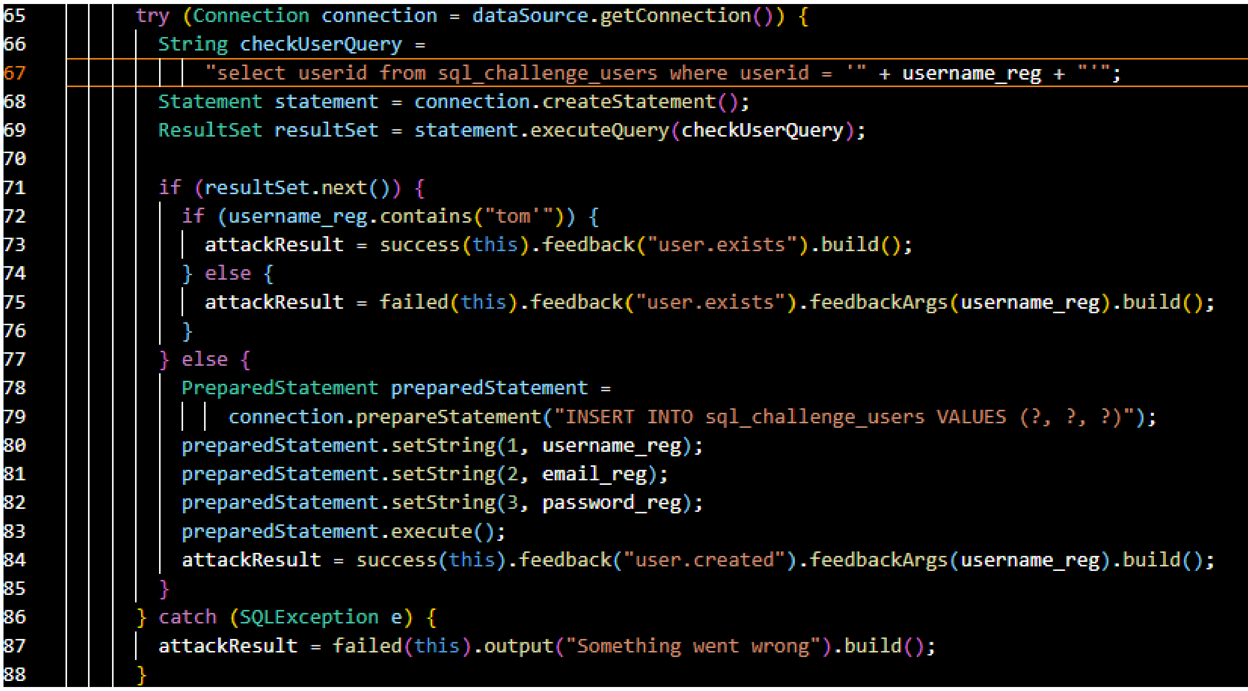


org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge

Bug	Category	Details	Line	Priority
new org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge(LessonDataSource) may expose internal representation by storing an externally mutable object into SqlInjectionChallenge.dataSource	MALICIOUS_CODE	EI_EXPOSE_REP2	50	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) may fail to clean up java.sql.ResultSet	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	69	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) may fail to clean up java.sql.Statement	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	68	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) may fail to close PreparedStatement	BAD_PRACTICE	ODR_OPEN_DATABASE_RESOURCE	79	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) may fail to close Statement	BAD_PRACTICE	ODR_OPEN_DATABASE_RESOURCE	68	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) passes a nonconstant String to an execute or addBatch method on an SQL statement	SECURITY	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	69	High

b. Bug Description:

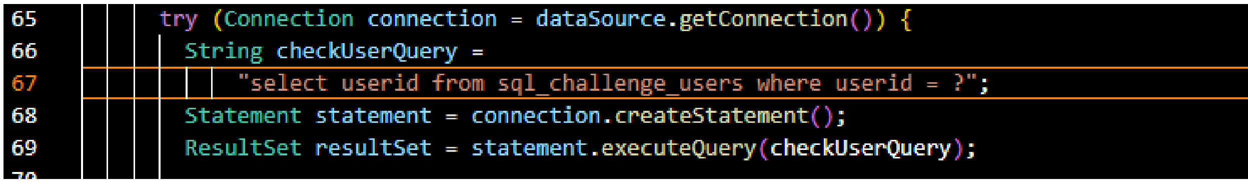
This error is very similar to the error found in Bug #1. Once again a prepared statement should be used to reduce the risks of SQL injection. The username_reg variable being concatenated within the checkUserQuery String is the cause of the bug.



```
try (Connection connection = dataSource.getConnection()) {
  String checkUserQuery =
      "select userid from sql_challenge_users where userid = '" + username_reg + "'";
  Statement statement = connection.createStatement();
  ResultSet resultSet = statement.executeQuery(checkUserQuery);

  if (resultSet.next()) {
    if (username_reg.contains("tom'")) {
      attackResult = success(this).feedback("user.exists").build();
    } else {
      attackResult = failed(this).feedback("user.exists").feedbackArgs(username_reg).build();
    }
  } else {
    PreparedStatement preparedStatement =
        connection.prepareStatement("INSERT INTO sql_challenge_users VALUES (?, ?, ?)");
    preparedStatement.setString(1, username_reg);
    preparedStatement.setString(2, email_reg);
    preparedStatement.setString(3, password_reg);
    preparedStatement.execute();
    attackResult = success(this).feedback("user.created").feedbackArgs(username_reg).build();
  }
} catch (SQLException e) {
  attackResult = failed(this).output("Something went wrong").build();
}
```

c. Edited Code:

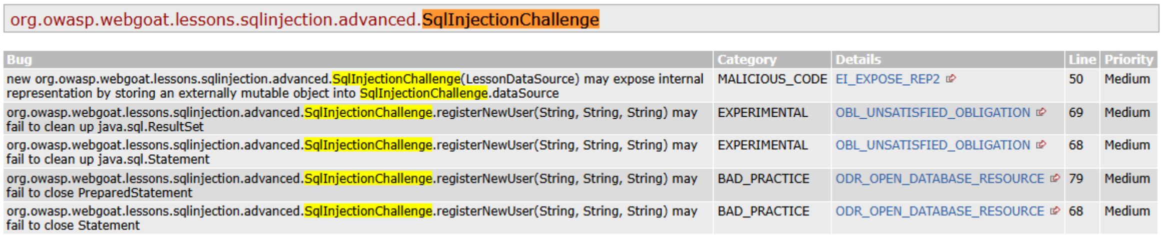
Removing the username_reg variable from within the checkUserQuery string will reduce the risks of sql injection.



```
try (Connection connection = dataSource.getConnection()) {
  String checkUserQuery =
      "select userid from sql_challenge_users where userid = ?";
  Statement statement = connection.createStatement();
  ResultSet resultSet = statement.executeQuery(checkUserQuery);
```

d. Updated Bug Report

There are still several other bugs that remain from this code. However, I wanted to show that changing this one line would remove the Security category bug from the report alone.

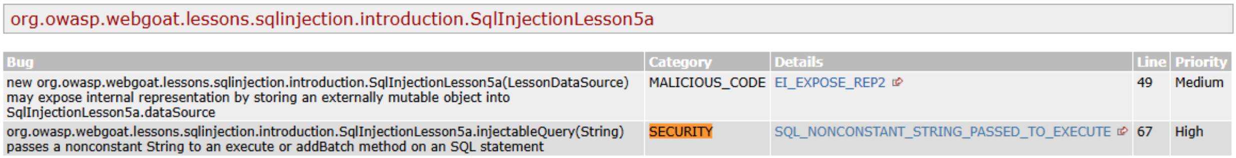


org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge

Bug	Category	Details	Line	Priority
new org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge(LessonDataSource) may expose internal representation by storing an externally mutable object into SqlInjectionChallenge.dataSource	MALICIOUS_CODE	EI_EXPOSE_REP2	50	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) may fail to clean up java.sql.ResultSet	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	69	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) may fail to clean up java.sql.Statement	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	68	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) may fail to close PreparedStatement	BAD_PRACTICE	ODR_OPEN_DATABASE_RESOURCE	79	Medium
org.owasp.webgoat.lessons.sqlinjection.advanced.SqlInjectionChallenge.registerNewUser(String, String, String) may fail to close Statement	BAD_PRACTICE	ODR_OPEN_DATABASE_RESOURCE	68	Medium

3. Bug #3

a. Initial Bug Report

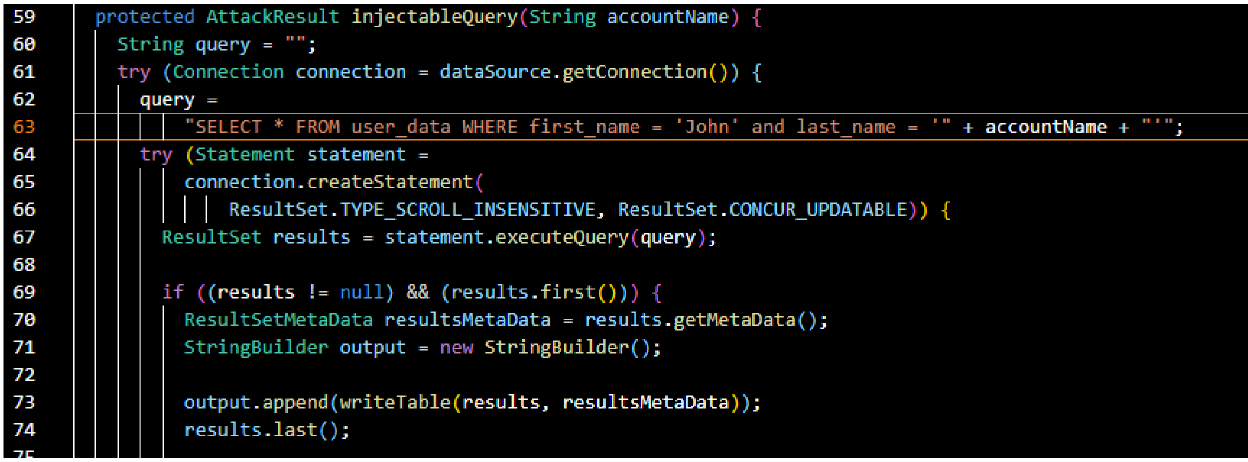


org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson5a

Bug	Category	Details	Line	Priority
new org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson5a(LessonDataSource) may expose internal representation by storing an externally mutable object into SqlInjectionLesson5a.dataSource	MALICIOUS_CODE	EI_EXPOSE_REP2	49	Medium
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson5a.injectableQuery(String) passes a nonconstant String to an execute or addBatch method on an SQL statement	SECURITY	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	67	High

b. Bug Description:

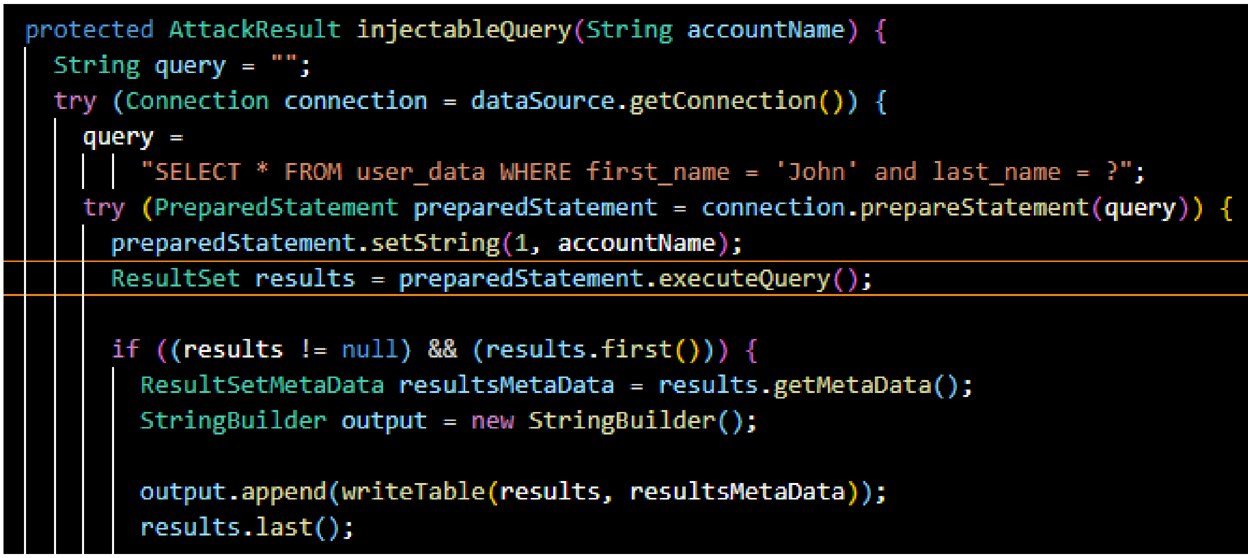
There is a nonconstant string passed to execute on line 67 of the code on SqlInjectionLesson5a. The accountName variable is concatenated to the query string and opens a vulnerability for an SQL injection.



```
59  protected AttackResult injectableQuery(String accountName) {
60    String query = "";
61    try (Connection connection = dataSource.getConnection()) {
62      query =
63          "SELECT * FROM user_data WHERE first_name = 'John' and last_name = '" + accountName + "'";
64      try (Statement statement =
65          connection.createStatement(
66              ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE)) {
67        ResultSet results = statement.executeQuery(query);
68
69        if ((results != null) && (results.first())) {
70          ResultSetMetaData resultsMetaData = results.getMetaData();
71          StringBuilder output = new StringBuilder();
72
73          output.append(writeTable(results, resultsMetaData));
74          results.last();
```

c. Edited Code:

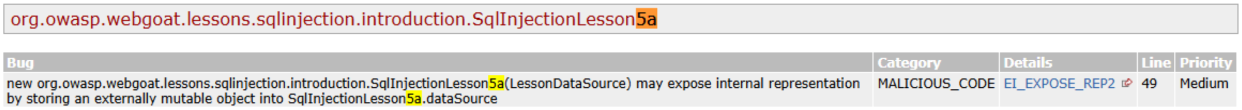
In the edited code, the '?' is used as a placeholder to ensure that only data passes through and not an outside (malicious) query.



```java
protected AttackResult injectableQuery(String accountName) {
  String query = "";
  try (Connection connection = dataSource.getConnection()) {
    query =
        "SELECT * FROM user_data WHERE first_name = 'John' and last_name = ?";
    try (PreparedStatement preparedStatement = connection.prepareStatement(query)) {
      preparedStatement.setString(1, accountName);
      ResultSet results = preparedStatement.executeQuery();

      if ((results != null) && (results.first())) {
        ResultSetMetaData resultsMetaData = results.getMetaData();
        StringBuilder output = new StringBuilder();

        output.append(writeTable(results, resultsMetaData));
        results.last();
```

d. Updated Bug Report

The security bug is now gone under the updated spotbugs report. Only one of the two initial bugs remains in the report.

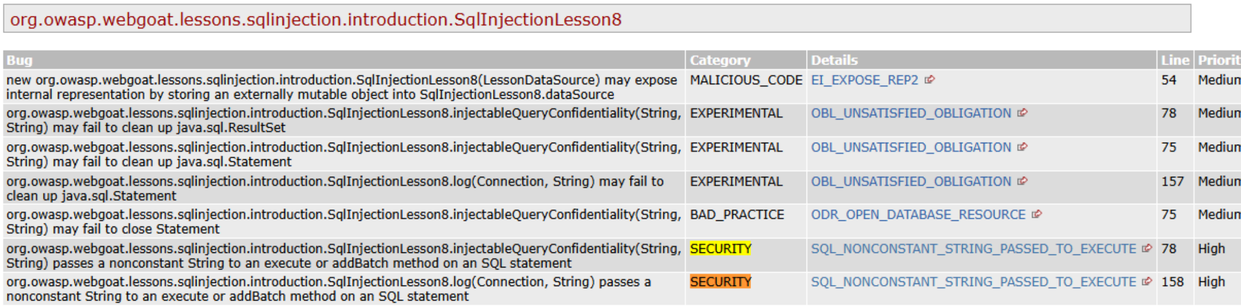


org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson5a

Bug	Category	Details	Line	Priority
new org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson5a(LessonDataSource) may expose internal representation by storing an externally mutable object into SqlInjectionLesson5a.dataSource	MALICIOUS_CODE	EI_EXPOSE_REP2	49	Medium

4. Bug #4

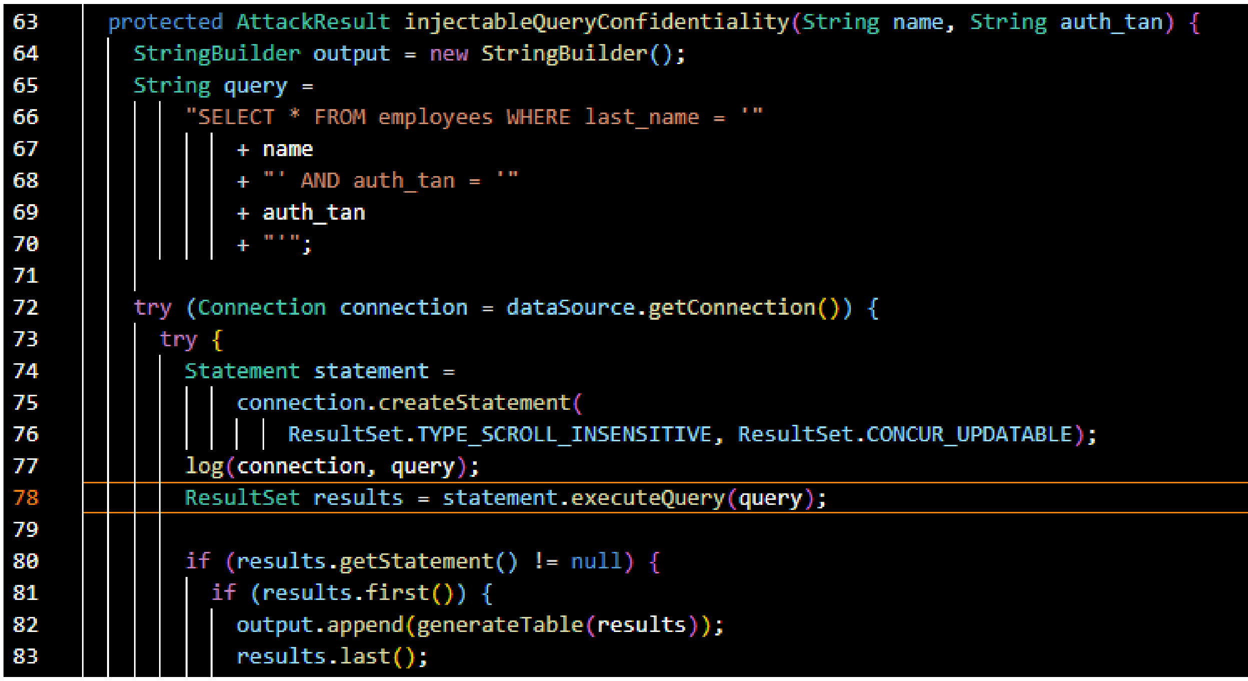
a. Initial Bug Report



org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8

Bug	Category	Details	Line	Priorit
new org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8(LessonDataSource) may expose internal representation by storing an externally mutable object into SqlInjectionLesson8.dataSource	MALICIOUS_CODE	EI_EXPOSE_REP2	54	Mediun
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.injectableQueryConfidentiality(String, String) may fail to clean up java.sql.ResultSet	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	78	Mediun
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.injectableQueryConfidentiality(String, String) may fail to clean up java.sql.Statement	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	75	Mediun
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.log(Connection, String) may fail to clean up java.sql.Statement	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	157	Mediun
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.injectableQueryConfidentiality(String, String) may fail to close Statement	BAD_PRACTICE	ODR_OPEN_DATABASE_RESOURCE	75	Mediun
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.injectableQueryConfidentiality(String, String) passes a nonconstant String to an execute or addBatch method on an SQL statement	SECURITY	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	78	High
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.log(Connection, String) passes a nonconstant String to an execute or addBatch method on an SQL statement	SECURITY	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	158	High

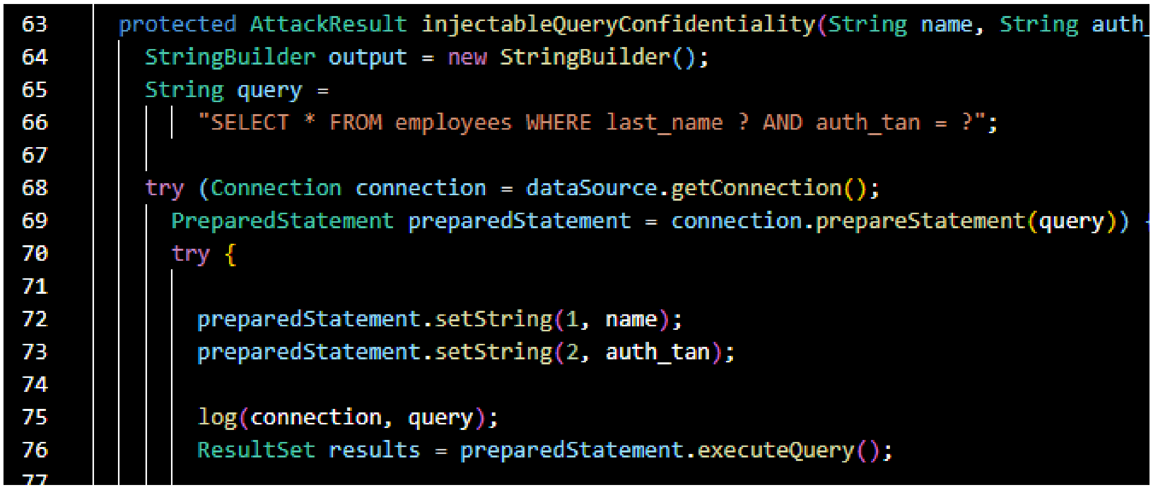
b. **Bug Description:** This bug is a nonconstant string passed to execute type. However, this time there are two different variables concatenated to the query string. This would make it potentially even easier to execute an SQL injection as there is now another field that can be modified to a statement such as “1 = 1”.



```java
protected AttackResult injectableQueryConfidentiality(String name, String auth_tan) {
  StringBuilder output = new StringBuilder();
  String query =
      "SELECT * FROM employees WHERE last_name = '"
          + name
          + "' AND auth_tan = '"
          + auth_tan
          + "'";

  try (Connection connection = dataSource.getConnection()) {
    try {
      Statement statement =
          connection.createStatement(
              ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
      log(connection, query);
      ResultSet results = statement.executeQuery(query);

      if (results.getStatement() != null) {
        if (results.first()) {
          output.append(generateTable(results));
          results.last();
```

c. Edited Code

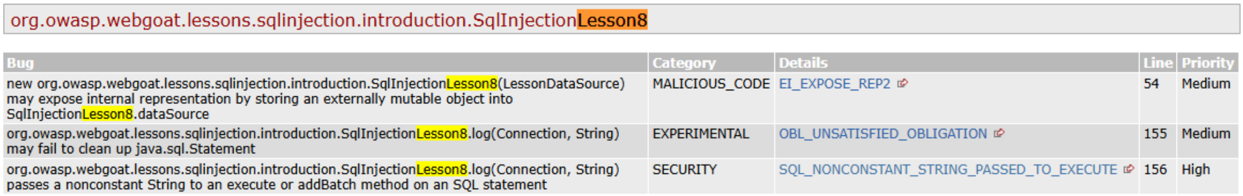
I removed the variables(name and auth_tan) from the query string and replaced them with ‘?’ placeholders. I then more safely set the parameters from within the try statement.



```java
protected AttackResult injectableQueryConfidentiality(String name, String auth_
  StringBuilder output = new StringBuilder();
  String query =
      "SELECT * FROM employees WHERE last_name ? AND auth_tan = ?";

  try (Connection connection = dataSource.getConnection();
    PreparedStatement preparedStatement = connection.prepareStatement(query))
    try {

      preparedStatement.setString(1, name);
      preparedStatement.setString(2, auth_tan);

      log(connection, query);
      ResultSet results = preparedStatement.executeQuery();
```

d. Updated Bug Report

After recompiling and getting a new report, there is now only one bug in the security category remaining on the Lesson 8. The next one will be taken care of on the final question.

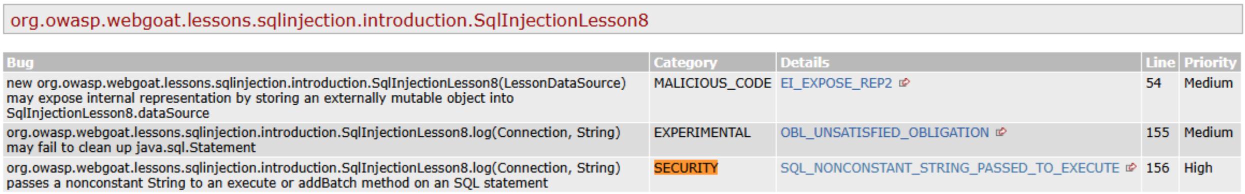


org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8

Bug	Category	Details	Line	Priority
new org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8(LessonDataSource) may expose internal representation by storing an externally mutable object into SqlInjectionLesson8.dataSource	MALICIOUS_CODE	EI_EXPOSE_REP2	54	Medium
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.log(Connection, String) may fail to clean up java.sql.Statement	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	155	Medium
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.log(Connection, String) passes a nonconstant String to an execute or addBatch method on an SQL statement	SECURITY	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	156	High

5. Bug #5

a. Initial Bug Report

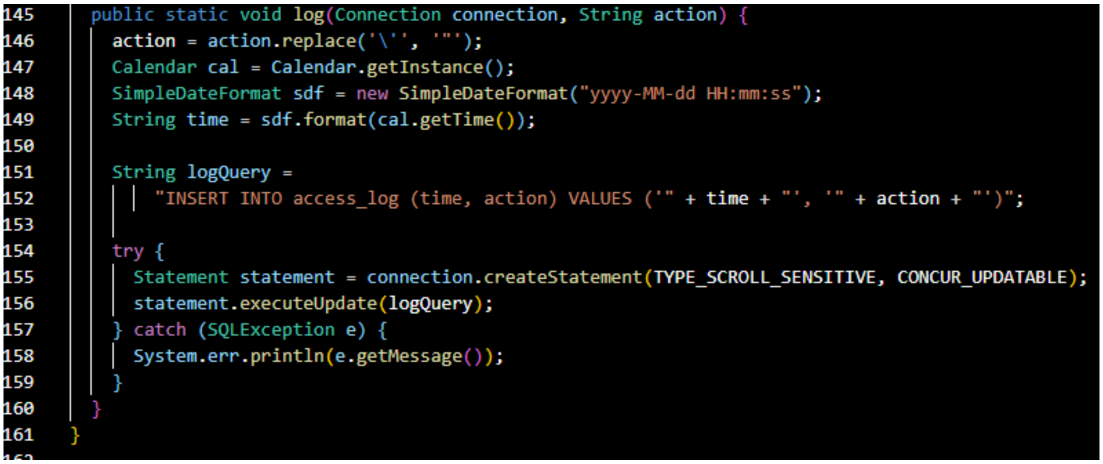


org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8

Bug	Category	Details	Line	Priority
new org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8(LessonDataSource) may expose internal representation by storing an externally mutable object into SqlInjectionLesson8.dataSource	MALICIOUS_CODE	EI_EXPOSE_REP2	54	Medium
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.log(Connection, String) may fail to clean up java.sql.Statement	EXPERIMENTAL	OBL_UNSATISFIED_OBLIGATION	155	Medium
org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8.log(Connection, String) passes a nonconstant String to an execute or addBatch method on an SQL statement	SECURITY	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	156	High

b. Bug Description:

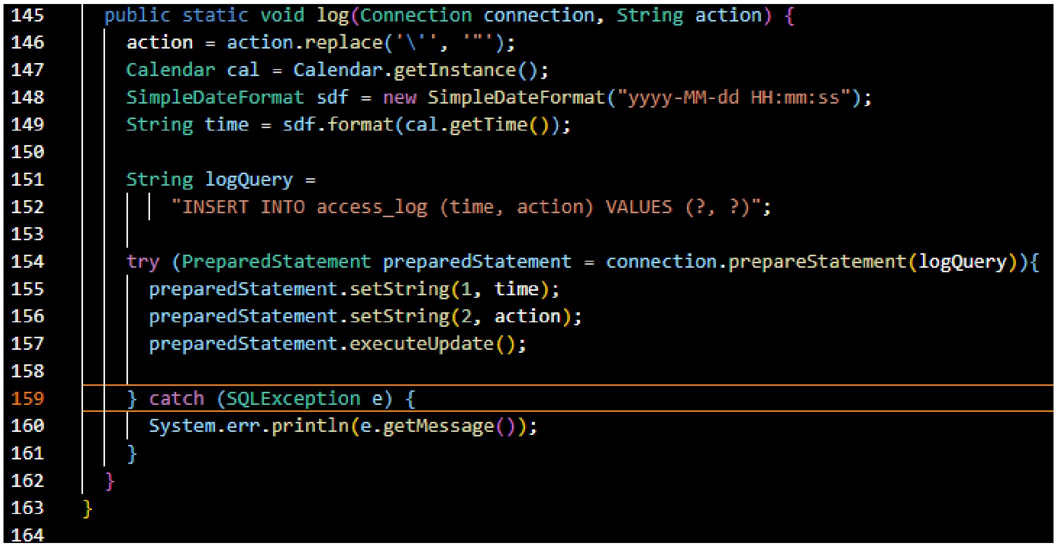
This is the second bug of the security category within lesson8. This string calls for two variables within parentheses. The two variables are the time and action variables. This dynamic string creates an opportunity for an SQL injection and could lead to loss of confidentiality and integrity within the accessed database.



```java
public static void log(Connection connection, String action) {
    action = action.replace('\'', '"');
    Calendar cal = Calendar.getInstance();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String time = sdf.format(cal.getTime());

    String logQuery =
        "INSERT INTO access_log (time, action) VALUES ('" + time + "', '" + action + "')";

    try {
      Statement statement = connection.createStatement(TYPE_SCROLL_SENSITIVE, CONCUR_UPDATABLE);
      statement.executeUpdate(logQuery);
    } catch (SQLException e) {
      System.err.println(e.getMessage());
    }
  }
}
```

c. Edited Code

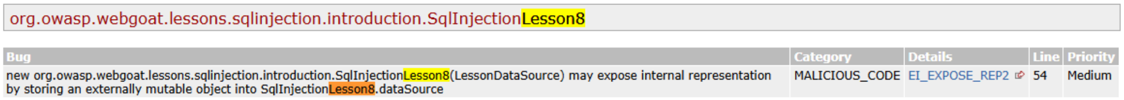
To fix this problem I replaced the values with placeholders '?'. This prevents unwanted values from being entered other than the data. This data is collected safely from the parameterized query below.



```java
public static void log(Connection connection, String action) {
    action = action.replace('\'', '"');
    Calendar cal = Calendar.getInstance();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String time = sdf.format(cal.getTime());

    String logQuery =
        "INSERT INTO access_log (time, action) VALUES (?, ?)";

    try (PreparedStatement preparedStatement = connection.prepareStatement(logQuery)){
      preparedStatement.setString(1, time);
      preparedStatement.setString(2, action);
      preparedStatement.executeUpdate();

    } catch (SQLException e) {
      System.err.println(e.getMessage());
    }
  }
}
```

d. Updated Bug Report

There are now not any security category bugs reported on the lesson 8 as seen below.

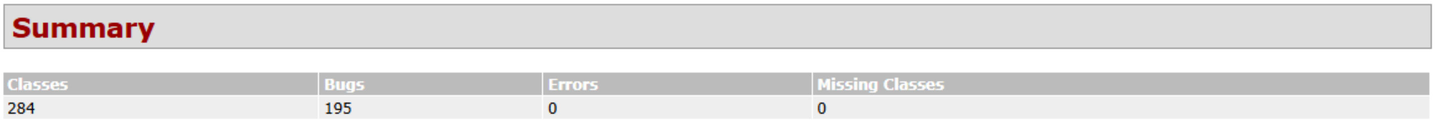


org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8

Bug	Category	Details	Line	Priority
new org.owasp.webgoat.lessons.sqlinjection.introduction.SqlInjectionLesson8(LessonDataSource) may expose internal representation by storing an externally mutable object into SqlInjectionLesson8.dataSource	MALICIOUS_CODE	EI_EXPOSE_REP2	54	Medium

Conclusion

In this lab, I utilized the spotbugs static analysis tool to identify security vulnerabilities from within the WebGoat application. All of the security vulnerabilities found within the program related to a nonconstant being passed into a query string. This opens the possibility of a SQL injection as a malicious user could enter an always true value such as “1 = 1” into the value position. This would compromise both the integrity and confidentiality of the data held within the database. However, thanks to the tools in this lab I was able to rectify 5 of the security vulnerabilities that included this error and also fix a few other less pressing issues in the process. In total, the total bug count went from 209 to 195 in just the 5 exercises held within the lab.



Summary

Classes	Bugs	Errors	Missing Classes
284	195	0	0