using NUnit.Framework;

using profdepo\_server.Extensions;

namespace Test\_RemoveSqlInjections

{

public class RemoveSqlInjections

{

[SetUp]

public void Setup()

{

}

// 1. Проверка удаления простых SQL-инъекций.

[TestCase("INSERT INTO Users VALUES ('John', 'Doe');", "")]

[TestCase("delete from Orders where UserId = 5;", "")]

[TestCase("DROP TABLE Customers;", "")]

[TestCase("SELECT \* FROM Products;", "")]

[TestCase("", "")]

[TestCase("SELECT \* FROM Users WHERE Name = 'John';", "")]

[TestCase("INSERT INTO Log (Message) VALUES ('Test');", "")]

[TestCase("DELETE FROM Products WHERE Category = 'Books';", "")]

[TestCase("DROP DATABASE Test;", "")]

[TestCase("SELECT \* FROM Customers;", "")]

[TestCase("UPDATE Users SET Name = 'John' WHERE Id = 1;", "")]

[TestCase("INSERT INTO Orders (ProductId, Quantity) VALUES (1, 10);", "")]

[TestCase("DELETE FROM Customers WHERE Country = 'USA';", "")]

[TestCase("ALTER TABLE Products ADD COLUMN Price decimal(10,2);", "")]

[TestCase("SELECT COUNT(\*) FROM Orders;", "")]

[TestCase("SELECT \* FROM Products WHERE Price > 100;", "")]

[TestCase("CREATE TABLE Employees (Id int, Name varchar(50));", "")]

[TestCase("SELECT AVG(Salary) FROM Employees WHERE Department = 'Sales';", "")]

// 2. Проверка защиты от UNION SQL-инъекций.

[TestCase("SELECT \* FROM Products WHERE Category = 'Books' UNION SELECT \* FROM Users;", "")]

[TestCase("SELECT \* FROM Products UNION SELECT \* FROM Users;", "")]

[TestCase("SELECT \* FROM Products WHERE Price > 100 UNION SELECT \* FROM Users WHERE 1=1--;", "")]

[TestCase("SELECT \* FROM Customers UNION SELECT \* FROM Users WHERE 1=0--;", "")]

[TestCase("SELECT \* FROM Products; SELECT \* FROM Users UNION SELECT \* FROM Log;", "")]

[TestCase("SELECT \* FROM Orders; DELETE FROM Customers UNION SELECT \* FROM Users WHERE 1=1--;", "")]

// 3. Проверка защиты от других SQL-инъекций.

[TestCase("'; SELECT \* FROM Users; --", "")]

[TestCase("1; DROP TABLE Orders; --", "")]

[TestCase("UPDATE Products SET Price = 0 WHERE 1=1--;", "")]

[TestCase("update Users SET Password = 'hacked' WHERE 1=0--;", "")]

[TestCase("UPDATE Customers SET Contact = 'John' WHERE Country = 'USA' OR 1=1--;", "")]

[TestCase("UPDATE Orders SET Status = 'Shipped' WHERE Quantity > 10 OR 1=0--;", "")]

[TestCase("SELECT \* FROM Products WHERE Category = 'Books' UNION SELECT \* FROM Users WHERE 1=1--;", "")]

[TestCase("UPDATE Products SET Price = Price \* 0.9 WHERE Category = 'Electronics' OR 1=1--;", "")]

// UNION SQL-инъекция с использованием подзапроса

[TestCase("SELECT \* FROM Products WHERE Category = 'Books' UNION SELECT \* FROM Users WHERE UserId = (SELECT UserId FROM Orders WHERE ProductId = 1);", "")]

// Использование временных таблиц

[TestCase("CREATE TABLE #TempTable (Id int, Name varchar(50)); INSERT INTO #TempTable VALUES (1, 'John'); SELECT \* FROM #TempTable;", "")]

// BLIND SQL-инъекция

[TestCase("SELECT \* FROM Users WHERE Id = 1; IF (1=1) WAITFOR DELAY '0:0:10'; --", "")]

// Многоуровневая SQL-инъекция

[TestCase("SELECT \* FROM Products WHERE Category = (SELECT Category FROM Categories WHERE CategoryId = '1');", "")]

// Time-Based SQL-инъекция

[TestCase("SELECT \* FROM Products WHERE Category = 'Books'; WAITFOR DELAY '0:0:10'; --", "")]

// Error-Based SQL-инъекция

[TestCase("SELECT \* FROM Products WHERE Category = 'Books' AND 1=CONVERT(int, (SELECT @@version)); --", "")]

// Boolean-Based SQL-инъекция

[TestCase("SELECT \* FROM Users WHERE Id = 1 OR (SELECT COUNT(\*) FROM Orders) > 0; --", "")]

// Out-of-Band SQL-инъекция

[TestCase("SELECT \* FROM Products; EXEC xp\_cmdshell('ping 10.10.10.10'); --", "")]

// Second-Order SQL-инъекция

[TestCase("INSERT INTO Orders (ProductId, Quantity) VALUES (1, 10); --", "")]

// Другие примеры SQL-инъекций (добавлены новые примеры):

// UNION SQL-инъекция с подменой возвращаемого количества столбцов

[TestCase("SELECT \* FROM Products WHERE Category = 'Books' UNION SELECT UserId, UserName, Password FROM Users; --", "")]

// Error-Based SQL-инъекция с извлечением информации через ошибки

[TestCase("SELECT \* FROM Products WHERE Category = 'Books' AND 1=(SELECT COUNT(\*) FROM NonExistentTable); --", "")]

// Использование комментариев для обхода фильтров

[TestCase("SELECT \* FROM Users WHERE Id = 1 /\* AND IsActive = 1 \*/;", "")]

// Внедрение SQL-кода в числовых значениях

[TestCase("SELECT \* FROM Products WHERE ProductId = 1 OR 1=1; --", "")]

// Использование обратных слешей для обхода фильтров

[TestCase("SELECT \* FROM Users WHERE Name = 'John'\' OR 1=1;", "")]

// Использование Unicode-символов для обхода фильтров

[TestCase("SELECT \* FROM Users WHERE Name = N'John' OR 1=1;", "")]

public void RemoveSqlInject(string input, string expectedOutput)

{

string value = ExtensionString.RemoveSqlInjections(input);

Assert.That(value, Is.EqualTo(expectedOutput));

}

}

}

**Чек-листы для проверки XSS, SQL- и HTML-инъекций**

В продолжение темы [чек-листов](https://kubig.blogspot.com/search/label/%D0%A7%D0%B5%D0%BA-%D0%BB%D0%B8%D1%81%D1%82) для [тестировщиков ПО](https://kubig.blogspot.com/search/label/Tester). Будете учиться, знаете где списать)

|  |  |
| --- | --- |
| **SQL-инъекции** | кавычка ' |
| OR '1'='1' |
| ' or '1'='1' -- ' |
| ' or '1'='1' ({ ' |
| ' or '1'='1' /\* ' |
| 12345) AND 1=1--:hash |
| SELECT user(); или SELECT system\_user(); |
| SELECT host, user, password FROM mysql.user; |
| SELECT database() |
| DROP TABLE user; |

|  |  |
| --- | --- |
| **XSS** | <script>alert("xss-injection!")</script> |
| <script>document.getElementByID("...").disabled=true</script> |
| <script>window.parent.location.href='http://hacker\_site';</script> |
| <img src=javascript:alert('xss-injection!')> |
| <input onclick="javascript:alert('xss-injection');"> |
| <b onmouseover="alert('xss-injection!')">Hello</b> |

|  |  |
| --- | --- |
| **HTML-инъекции** | </body> |
| <textarea /> |
| <input></input> |
| <form action="http://google.com"><input type="submit"></form> |