Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ

к лабораторной работе №6

на тему

**ЗАЩИТА ОТ АТАКИ МЕТОДОМ ВНЕДРЕНИЯ SQL-КОДА**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Герчик А.В. |

Минск 2025

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc189743996)

[2 Реализация программного средства 4](#_Toc189743997)

[Заключение 5](#_Toc189743998)

[Список использованных источников 6](#_Toc189743999)

[Приложение А (обязательное) Исходный код программы 7](#_Toc189744000)

**1 ПОСТАНОВКА ЗАДАЧИ**

В данной лабораторной работе будет проведено исследование уязвимостей веб-приложений, связанных с атакой методом внедрения SQL-кода, а также разработка эффективных методов защиты от таких атак. Задача состоит в создании тестовой базы данных, содержащей таблицу пользователей, и демонстрации потенциальных уязвимостей, возникающих при недостаточной обработке пользовательского ввода.

В рамках работы необходимо сконструировать тестовое веб-приложение с намеренно оставленными уязвимостями, что позволит продемонстрировать, как злоумышленники могут использовать недостатки в обработке данных для выполнения произвольных SQL-запросов. Основное внимание будет уделено анализу механизма атаки, который позволяет получать доступ к конфиденциальной информации, такой как учетные данные пользователей.

После демонстрации уязвимости следует перейти к разработке безопасных методов обработки пользовательского ввода, направленных на предотвращение успешных атак методом внедрения SQL-кода. В этом контексте будет рассмотрено использование параметризованных запросов и других подходов, которые позволяют отделять данные от кода, что значительно снижает риск выполнения вредоносных запросов.

В отчете необходимо отразить весь процесс работы, начиная с создания уязвимого приложения и заканчивая внедрением безопасных методов. Важно описать выявленные уязвимости, проведенные тесты и предложенные решения, а также оценить их эффективность. Завершая работу, следует обсудить значимость соблюдения принципов безопасности в разработке веб-приложений и необходимость постоянного обновления и мониторинга систем защиты.

**2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО СРЕДСТВА**

Для выполнения лабораторной работы разработали программное средство на языке Python, демонстрирующее уязвимость к SQL-инъекциям и методы защиты от них. Программа состояла из двух частей: уязвимой и защищенной реализации аутентификации пользователя.

Для тестирования была создана базу данных в памяти с использованием модуля sqlite3. В базе данных была создана таблица *users* с полями *id*, *username* и *password*. Добавили тестового пользователя с логином *admin* и паролем *secure\_password*.

Для тестирования создана функция *unsafe\_login*, которая формирует *SQL*-запрос с использованием конкатенации строк. Это сделало приложение уязвимым к *SQL*-инъекциям. Например, при вводе пароля «*OR* '1'='1'» злоумышленник мог обойти проверку аутентификации.

Для защиты от *SQL*-инъекций реализована функция *safe\_login*, использующую параметризованные запросы. В этом случае пользовательский ввод автоматически экранировался, что предотвращало возможность внедрения вредоносного *SQL*-кода.

Было проведено тестирование обеих реализаций (рисунок 2.1). В уязвимой версии при вводе пароля «OR '1'='1'» аутентификация проходила успешно, что подтвердило наличие уязвимости. В защищенной версии та же атака не сработала, и доступ был отклонен.

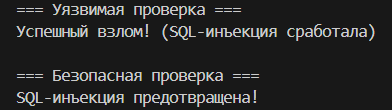


Рисунок 2.1 – Результат тестирования

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы успешно реализовали программное средство, демонстрирующее уязвимость к *SQL*-инъекциям и методы защиты от них. Создали тестовую базу данных с таблицей пользователей, добавив туда тестовые данные для проверки функциональности. Разработали уязвимую версию аутентификации, использующую конкатенацию строк для формирования *SQL*-запросов, что позволило продемонстрировать успешную *SQL*-инъекцию при вводе злоумышленником специально сформированных данных. Для защиты от подобных атак реализовали безопасную версию аутентификации, основанную на параметризованных запросах, которые автоматически экранируют пользовательский ввод, предотвращая внедрение вредоносного *SQL*-кода.

Тестирование программы подтвердило эффективность параметризованных запросов. В уязвимой версии при вводе строки «*OR* '1'='1'» аутентификация проходила успешно, что наглядно показало наличие уязвимости. В защищенной версии та же атака была предотвращена, и доступ к системе отклонялся. Это подтвердило, что использование параметризованных запросов является надежным методом защиты от *SQL*-инъекций.

Результаты работы подчеркнули важность соблюдения принципов безопасности при разработке приложений, работающих с базами данных. Неправильная обработка пользовательского ввода может привести к серьезным последствиям, включая утечку данных, несанкционированный доступ и повреждение информации. Использование параметризованных запросов, *ORM*-систем и других современных методов защиты позволяет минимизировать риски и обеспечить безопасность приложения [1].

В заключение можно отметить, что выполнение данной работы позволило не только изучить теоретические основы *SQL*-инъекций, но и получить практический опыт реализации защищенных приложений. Полученные знания и навыки могут быть применены в дальнейшей работе для создания безопасных и надежных программных решений.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] SQL-инъекции для самых маленьких [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/725134/.

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

import sqlite3

def create\_database():

"""Создание тестовой базы данных и таблицы пользователей"""

conn = sqlite3.connect(':memory:')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE users (

id INTEGER PRIMARY KEY,

username TEXT NOT NULL,

password TEXT NOT NULL

)

''')

# Добавляем тестового пользователя

cursor.execute('INSERT INTO users (username, password) VALUES (?, ?)',

('admin', 'secure\_password'))

conn.commit()

return conn

# Уязвимая функция

def unsafe\_login(conn, username, password):

cursor = conn.cursor()

# ОПАСНО: конкатенация строк напрямую

query = f"SELECT \* FROM users WHERE username = '{username}' AND password = '{password}'"

cursor.execute(query)

return cursor.fetchone() is not None

# Безопасная функция с параметризованным запросом

def safe\_login(conn, username, password):

cursor = conn.cursor()

# Безопасно: использование параметризованного запроса

cursor.execute('SELECT \* FROM users WHERE username = ? AND password = ?',

(username, password))

return cursor.fetchone() is not None

def demonstration():

conn = create\_database()

evil\_password = "' OR '1'='1"

print("=== Уязвимая проверка ===")

if unsafe\_login(conn, 'admin', evil\_password):

print("Успешный взлом! (SQL-инъекция сработала)")

else:

print("Вход отклонен")

print("\n=== Безопасная проверка ===")

if safe\_login(conn, 'admin', evil\_password):

print("Вход выполнен")

else:

print("SQL-инъекция предотвращена!")

if \_\_name\_\_ == "\_\_main\_\_":

demonstration()