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

ОТЧЁТ

по лабораторной работе

по дисциплине «Базы данных»

Тема: «Реализация базы данных в СУБД PostgreSQL».

Студент гр. 1304 _____ Мусаев А.И.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2023

Цель работы

Задание

Вариант - 16.

Пусть требуется создать программную систему, предназначенную для врачей и работников регистратуры поликлиники. Такая система должна хранить сведения об участках, которые относятся к поликлинике, о расписании работы участковых врачей, информацию о врачах, а также карточки пациентов. Карточка имеет номер, в нее заносятся сведения о каждом посещении поликлиники пациентом: дата посещения, жалобы, предварительный диагноз, назначения, выписан или нет больничный лист, и, если выписан, то на какой срок, имя врача. В карточке на первой странице указаны также фамилия, имя, отчество пациента, его домашний адрес, пол и возраст, номер страхового полиса, дата заполнения карточки. В расписании работы врачей указывается, на каком участке работает врач, дни и часы приема, номер кабинета. Врач может обслуживать более одного участка. В случае увольнения врача его участок(участки)передается другим врачам. Данные о враче, которые хранятся в БД, - это фамилия, имя отчество, категория, стаж работы, дата рождения. В карточку больного при каждом его посещении поликлиники врачом заносится очередная запись. Работники регистратуры регистрируют пациента, заполняя первую страницу его карточки. Уволить врача имеет право только заведующий поликлиникой. Он удаляет из базы сведения о враче и передает его больных другому врачу.

Работникам поликлиники могут потребоваться следующие сведения:

- Адрес данного больного, дата последнего посещения поликлиники и диагноз?
- Фамилия и инициалы лечащего врача данного больного?
- Номер кабинета, дни и часы приема данного врача?
- Больные, находящиеся в данный момент на лечении у данного врача(не истек срок больничного листа);

- Назначения врачей при указанном заболевании?
- Кто работает в данный момент в указанном кабинете?
- Сколько раз за прошедший месяц обращался в поликлинику указанный больной?
- Адрес данного больного

Задачи:

1. Сделать простой web-сервер для выполнения запросов из ЛР3, например с express.js (flask). Не обязательно делать авторизацию и т.п., хватит одного эдпоинта на каждый запрос, с параметрами запроса как query parameters.
2. Намеренно сделайте несколько (2-3) запроса, подверженных SQL-инъекциям.
3. Проверьте Ваше API с помощью sqlmap (или чего-то аналогичного), передав эндпоинты в качестве целей атаки. Посмотрите, какие уязвимости он нашёл (и не нашёл), опишите пути к исправлению.
4. +2 балла, если напишете эндпоинт с уязвимостью, которая не находится sqlmap-ом.

Выполнение работы

В файле create_db.py создан web-сервер с использованием модуля flask.

Проверим работоспособность:

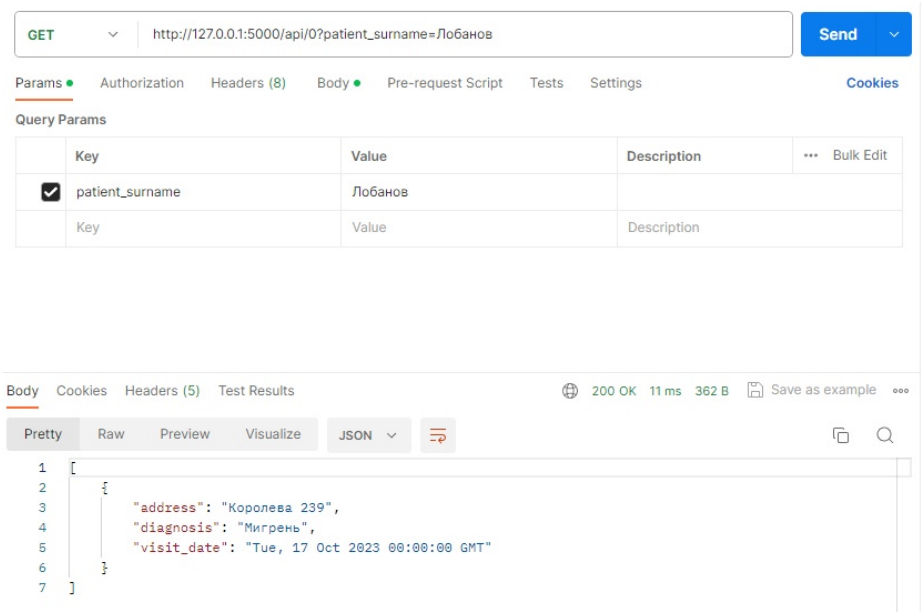

Рис. 1: Адрес данного больного, дата последнего посещения поликлиники и диагноз?

Остальные запросы тоже выдают желаемый результат.

Теперь запустим sql-map:

Для не инъекционного запроса:

Рис. 2: Фамилия и инициалы лечащего врача данного больного?

Поиск не дал результатов. Теперь попробуем для разных видов инъекционного запроса, один реализован через f-строку, другой - нет.

Рис. 3: Назначения врачей при указанном заболевании?

Это было тестирование f-строки, скорее всего, дело в выводе ret = [row_to_dict(row) for row in results], так вывод дает вывести только что-то в виде массива, а sql-map для тестирования, скорее всего, нужна другая информация, поэтому исполнение приводит к ошибке.

Тестирование другого запроса приводит к тому же результату.

Однако, запрос с ошибкой все же можно составить:

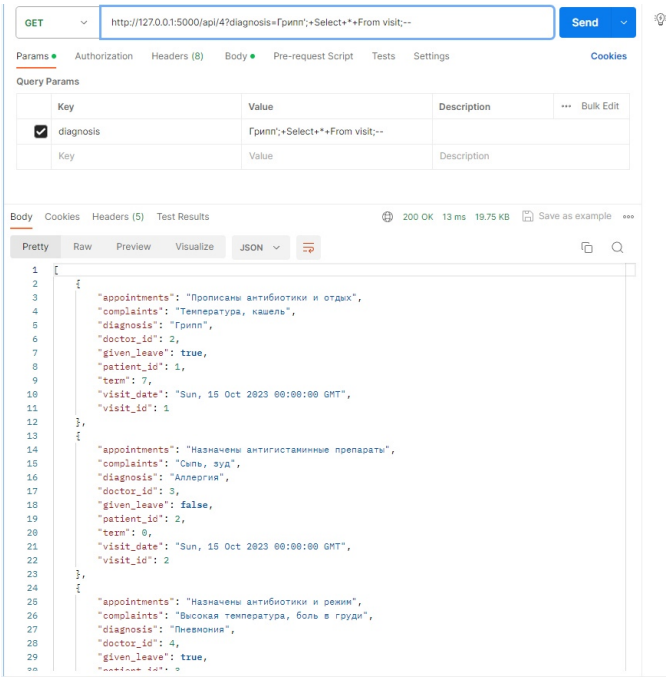

Рис. 4: Назначения врачей при указанном заболевании?

Таким образом, мы можем вытянуть из таблицы любые данные.

Вывод

В ходе выполнения работы реализован простой web-сервер для работы с базой данных и передачей параметров через параметры зарпоса. Два запроса изменены, чтобы быть подверженными SQL-инъекциям. Проведено тестирование базы данных с помощью sqlmap, внесенные уязвимости были обнаружены.

Приложение А

Ссылки:

PR

Приложение В

Исходный код:

create_db.py:

```
from flask import Flask, request
from queris import queries_list


app = Flask(__name__)


@app.route("/api/<query_index>")
def index(query_index):
    if not query_index.isdigit() or int(query_index) > len(queries_list) - 1:
        return "Unexpected query ID"

    return queries_list[int(query_index)](request.args)


if __name__ == "__main__":
    app.run(debug=True)
```

queris.py:

```
from datetime import date
from sqlalchemy import create_engine, desc, func, sql
from sqlalchemy.orm import sessionmaker, class_mapper
from main import clinic, selection_clinic, doctor, doctor_schedule, patient, visit
from add import adding

engine = create_engine("postgresql+psycopg2://postgres:1111@localhost/lab_5_db")
session_class = sessionmaker(bind=engine)
session = session_class()

adding()

def row_to_dict(row):
    result = {}
```

```
    for column in row._fields:
        value = getattr(row, column)
        if isinstance(value, (list, dict, tuple)):
            value = row_to_dict(value)
        result[column] = value
    return result


def proceed_query_1(request_arguments):
    selected_patient_surname = request_arguments.get("patient_surname")

    if selected_patient_surname is None:
        return "Undefined"

    query_text = """
        SELECT patient.address, visit.visit_date, visit.diagnosis
        FROM patient
        JOIN visit on visit.patient_id = patient.patient_id
        WHERE patient.surname = :selected_patient_surname
        LIMIT 1;
    """
    results = session.execute(sql.text(query_text), {"selected_patient_surname": selected_patient_surname})
    result_dicts = [row_to_dict(row) for row in results]


    return result_dicts


def proceed_query_2(request_arguments):
    selected_patient_surname = request_arguments.get("patient_surname")

    if selected_patient_surname is None:
        return "Undefined"

    query_2 = session.query(func.concat(doctor.surname, ' ', func.substring(doctor.name, 1, 1), '.', ' ',
                                        func.substring(doctor.patronymic, 1, 1), '.').label("doctor_name"),
                            visit.diagnosis.label("diagnosis"),
                            visit.visit_date + visit.term.label("end_date")) \
        .join(visit, doctor.doctor_id == visit.doctor_id) \
        .join(patient, visit.patient_id == patient.patient_id) \
        .filter(patient.surname == selected_patient_surname)
    results = query_2.all()
    result_dicts = [row_to_dict(row) for row in results]
```

```
    return result_dicts

def proceed_query_3(request_arguments):
    selected_doctor_surname = request_arguments.get("doctor_surname")

    if selected_doctor_surname is None:
        return "Undefined"

    query_text = """
        SELECT DISTINCT doctor_schedule.cabinet, doctor_schedule.reception_date
        FROM doctor
        JOIN doctor_schedule on doctor_schedule.doctor_id = doctor.doctor_id
        WHERE doctor.surname = :selected_doctor_surname
    """
    results = session.execute(sql.text(query_text), {"selected_doctor_surname": selected_doctor_surname})
    result_dicts = [row_to_dict(row) for row in results]

    return result_dicts


def proceed_query_4(request_arguments):
    selected_doctor_surname = request_arguments.get("doctor_surname")

    if selected_doctor_surname is None:
        return "Undefined"

    query_4 = session.query(patient.surname, patient.name, patient.patronymic,
                            visit.visit_date, visit.diagnosis, visit.appointments, visit.complaints) \
        .join(visit, patient.patient_id == visit.patient_id) \
        .join(doctor, visit.doctor_id == doctor.doctor_id) \
        .filter(doctor.surname == selected_doctor_surname, (visit.visit_date + visit.term) >= date(2023, 1
    results = query_4.all()
    result_dicts = [row_to_dict(row) for row in results]

    return result_dicts

def proceed_query_5(request_arguments):
    selected_diagnosis = request_arguments.get("diagnosis")

    if selected_diagnosis is None:
        return "Undefined"
```

```
    try:
        results = session.execute(sql.text(f"SELECT appointments FROM visit WHERE diagnosis = '{selected_di
        ret = [row_to_dict(row) for row in results]
        return ret
    except Exception as e:
        print(f"Error: {e}")
        return "Internal error"


def proceed_query_6(request_arguments):
    selected_area = request_arguments.get("area")
    selected_cab = request_arguments.get("cab")

    if selected_area is None:
        return "Undefined"

    if selected_cab is None:
            return "Undefined"


    query_6 = session.query(doctor.surname, doctor.name, doctor.patronymic, doctor_schedule.cabinet) \
        .join(doctor_schedule, doctor.doctor_id == doctor_schedule.doctor_id) \
        .filter(doctor_schedule.reception_date == date(2023, 10, 23),
                doctor_schedule.name_area == selected_area,
                doctor_schedule.cabinet == selected_cab)
    results= query_6.all()
    result_dicts = [row_to_dict(row) for row in results]
    return result_dicts


def proceed_query_7(request_arguments):
    query_7 = session.query(doctor.surname, doctor.name, doctor.patronymic,
                            func.count(visit.visit_id).label("numPatients")) \
        .join(doctor_schedule, doctor.doctor_id == doctor_schedule.doctor_id) \
        .join(visit, doctor_schedule.schedule_id == visit.doctor_id) \
        .filter(visit.visit_date >= date(2023, 10, 1), visit.visit_date <= date(2023, 10, 31)) \
        .group_by(doctor.surname, doctor.name, doctor.patronymic) \
        .order_by(desc("numPatients"))
    results = query_7.all()
    result_dicts = [row_to_dict(row) for row in results]
    return result_dicts

def proceed_query_8(request_arguments):
```

```
    selected_patient_surname = request_arguments.get("patient_surname")

    if selected_patient_surname is None:
        return "Undefined"

    results = session.execute(sql.text(f"SELECT patient.address FROM patient WHERE patient.surname = '{sel
    print(results)
    result_dicts = [row_to_dict(row) for row in results]

    return result_dicts



queries_list = [proceed_query_1, proceed_query_2, proceed_query_3, proceed_query_4, proceed_query_5, proce
```