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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Базы данных»

Тема: Тестирование БД на безопасность

Студент гр. 1303 _____ Депрейс А.С.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2023

Цель работы.

Сделать простой web-сервер и проверить реализованное API на безопасность.

Текст задания

Вариант 7

1. Сделать простой web-сервер для выполнения запросов из ЛР3, например с (express.js). Не обязательно делать авторизацию и т.п., хватит одного эндпоинта 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.

Выполнение работы

Для выполнения работы были использованы модели из лабораторной работы №3. Реализовано API для доступа к БД.

Рисунок 1. – Реализованный web-сервер для выполнения запросов.

```
@Get( path: 'third')
async getThird(
  @Query() query: { age: string; dietNumber: string },
  @Res() response: Response,
) : Promise<void>  {
  try {
    response.json(
      await this.appService.getThird(query.age, query.dietNumber),
    );
  } catch (e) {
    response.status( code: 500).end();
  }
}

no usages
@Get( path: 'fourth')
async getFourth(
  @Query() query: { workerId: string },
  @Res() response: Response,
) : Promise<void>  {
  try {
    response.json(await this.appService.getFourth(parseInt(query.workerId)));
  } catch (e) {
    response.status( code: 500).end();
  }
}
```

Рисунок 2 – Реализованный web-сервер для выполнения запросов.

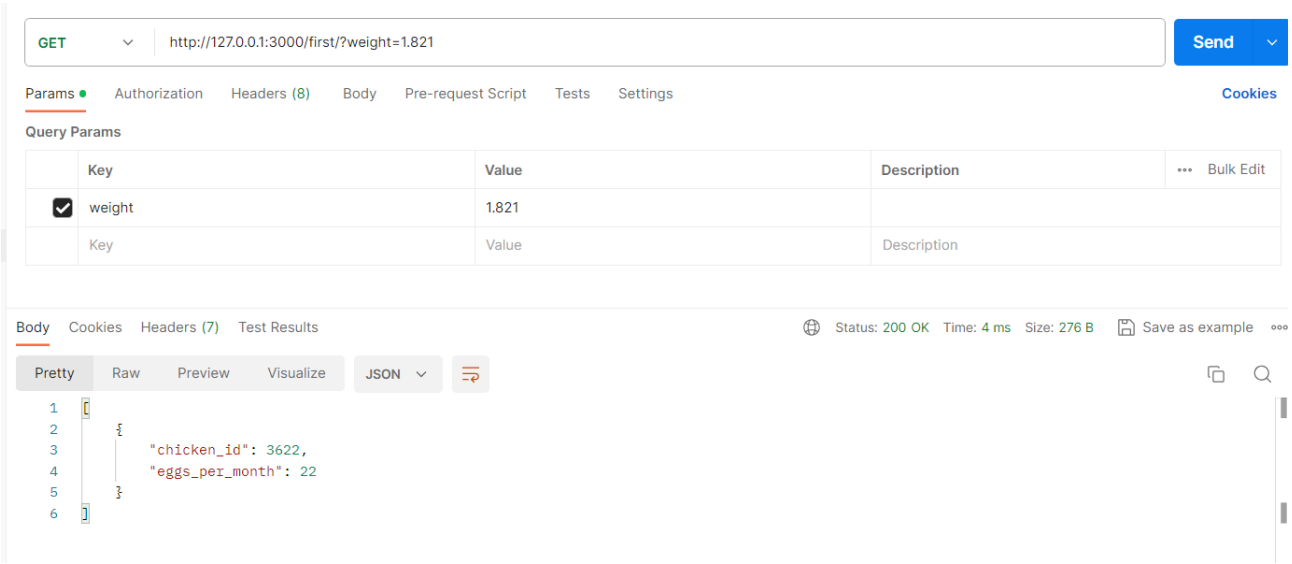


Рисунок 3. – Пример ответа web-сервера.

Второй и третий эндпоинты подвержены SQL-инъекциям:

```
async getSecond(breedName: string) : Promise<[?[], ?]> {
  return this.sequelize.query(
    sql: 'SELECT workshop_number FROM cell\n' +
      '\tINNER JOIN watching_chicken_in_cage USING(cell_id)\n' +
      '\tINNER JOIN chicken USING(chicken_id)\n' +
      'WHERE\n' +
      "\tbreed_name = '" + breedName + "'\n" +
      'GROUP BY workshop_number\n' +
      'ORDER BY COUNT(workshop_number) DESC\n' +
      'LIMIT 1;\n',
  );
}

1 usage
async getThird(age: string, dietNumber: string) : Promise<[?[], ?]> {
  return this.sequelize.query(
    sql: 'SELECT cell_id, workshop_number, row_number, cell_number FROM cell \n' +
      '\tINNER JOIN watching_chicken_in_cage USING(cell_id) \n' +
      '\tINNER JOIN chicken USING(chicken_id)\n' +
      '\tINNER JOIN breed USING(breed_name)\n' +
      'WHERE\n' +
      "\trecommended_diet_number = '" + dietNumber + "' AND" +
      "\tage = '" + age + "';",
  );
}
```

Рисунок 4. – Эндпоинты 2 и 3.

Первый и четвертый эндпоинты не подвержены SQL-инъекциям:

```
async getFirst(weight: number): Promise<Chicken[]> {
  return this.chickenRepository.findAll( options: {
    attributes: ['chicken_id', 'eggs_per_month'],
    where: {
      weight: {
        [Op.and]: {
          [Op.lt]: weight + 0.0001,
          [Op.gt]: weight - 0.0001,
        },
      },
    },
    limit: 10,
    order: [['eggs_per_month', 'DESC']],
  });
}
```

Рисунок 5. – Эндпоинт 1.

Рисунок 6. – Эндпоинт 4.

Проверим API с помощью sqlmap:

1. `python sqlmap.py -u "http://localhost:3000/first?weight=1.8" -p "weight"`

```
[20:02:15] [INFO] testing connection to the target URL
[20:02:16] [INFO] testing if the target URL content is stable
[20:02:16] [INFO] target URL content is stable
[20:02:16] [WARNING] heuristic (basic) test shows that GET parameter 'weight' might not be injectable
[20:02:16] [INFO] testing for SQL injection on GET parameter 'weight'
[20:02:16] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[20:02:16] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[20:02:16] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[20:02:16] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[20:02:16] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[20:02:16] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[20:02:16] [INFO] testing 'Generic inline queries'
[20:02:16] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[20:02:16] [WARNING] time-based comparison requires larger statistical model, please wait. (done)
[20:02:16] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[20:02:16] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[20:02:16] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[20:02:16] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[20:02:16] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[20:02:16] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number
of requests? [Y/n] y
[20:02:20] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[20:02:20] [WARNING] GET parameter 'weight' does not seem to be injectable
[20:02:20] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perf
orm more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g.
 '--tamper=space2comment') and/or switch '--random-agent'
```

Параметр weight не подвержено инъекции, так как если передать не число, то parseInt отдаст NaN и тогда запрос будет “корректным”.

2. `python sqlmap.py -u "http://127.0.0.1:3000/second/?breedName=soupy bad short-term" -p "breedName"`

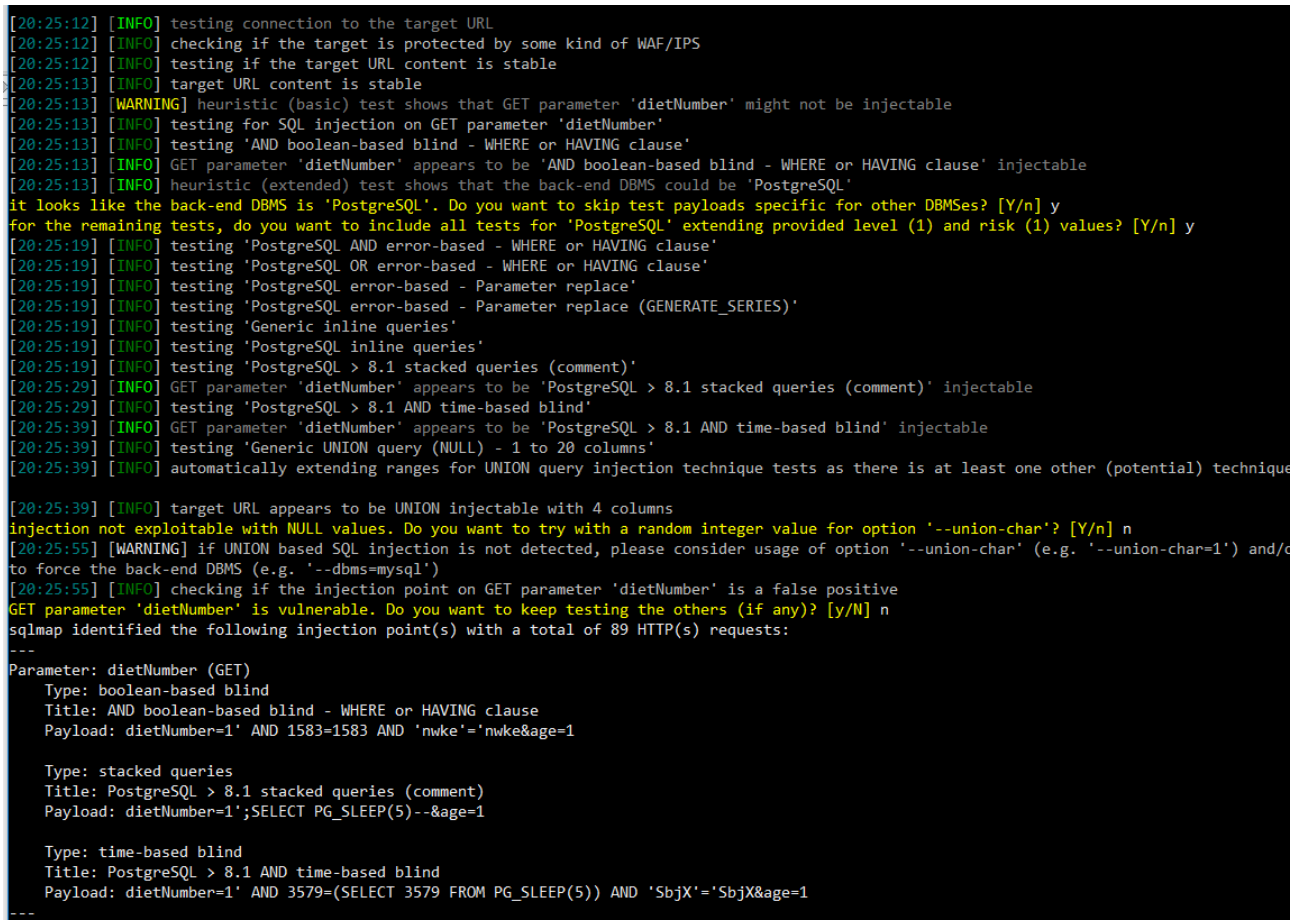
```
[20:09:10] [INFO] testing connection to the target URL
[20:09:10] [INFO] checking if the target is protected by some kind of WAF/IPS
[20:09:10] [INFO] testing if the target URL content is stable
[20:09:10] [INFO] target URL content is stable
[20:09:10] [WARNING] heuristic (basic) test shows that GET parameter 'breedName' might not be injectable
[20:09:10] [INFO] testing for SQL injection on GET parameter 'breedName'
[20:09:10] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[20:09:10] [INFO] GET parameter 'breedName' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[20:09:10] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'PostgreSQL'
it looks like the back-end DBMS is 'PostgreSQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'PostgreSQL' extending provided level (1) and risk (1) values? [Y/n] y
[20:09:23] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[20:09:23] [INFO] testing 'PostgreSQL OR error-based - WHERE or HAVING clause'
[20:09:23] [INFO] testing 'PostgreSQL error-based - Parameter replace'
[20:09:23] [INFO] testing 'PostgreSQL error-based - Parameter replace (GENERATE_SERIES)'
[20:09:23] [INFO] testing 'Generic inline queries'
[20:09:23] [INFO] testing 'PostgreSQL inline queries'
[20:09:23] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[20:09:23] [INFO] testing 'PostgreSQL > 8.1 stacked queries'
[20:09:23] [INFO] testing 'PostgreSQL stacked queries (heavy query - comment)'
[20:09:23] [INFO] testing 'PostgreSQL stacked queries (heavy query)'
[20:09:23] [INFO] testing 'PostgreSQL < 8.2 stacked queries (Glibc - comment)'
[20:09:23] [INFO] testing 'PostgreSQL < 8.2 stacked queries (Glibc)'
[20:09:23] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[20:09:33] [INFO] GET parameter 'breedName' appears to be 'PostgreSQL > 8.1 AND time-based blind' injectable
[20:09:33] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[20:09:33] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) t

[20:09:33] [INFO] checking if the injection point on GET parameter 'breedName' is a false positive
GET parameter 'breedName' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 74 HTTP(s) requests:
---
Parameter: breedName (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: breedName=soupy bad short-term' AND 7559=7559 AND 'CDom'='CDom

    Type: time-based blind
    Title: PostgreSQL > 8.1 AND time-based blind
    Payload: breedName=soupy bad short-term' AND 7345=(SELECT 7345 FROM PG_SLEEP(5)) AND 'pBdt'='pBdt
---
```

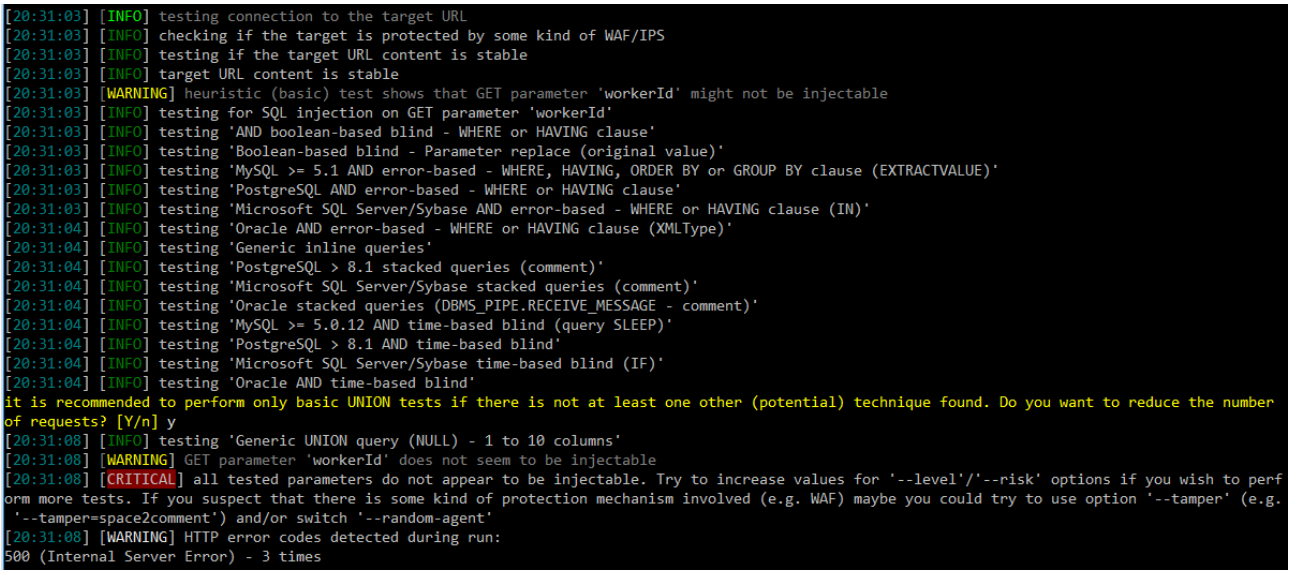
Параметр breedName подвержено SQL инъекциям boolean-based blind и time-based blind типа. Первое позволяет получить информацию из БД с помощью реакций на логические выражения в инъекции, а вторая позволяет получить информацию из БД в случае если нету видимого вывода, при успешной инъекции БД отвечает с задержкой.

3. `python sqlmap.py -u "http://127.0.0.1:3000/third/?dietNumber=1&age=1" -p "dietNumber,age"`



Параметр dietNumber подвержено SQL инъекциям. Кроме уже встречающихся инъекций появилась stacked queries инъекция, которая позволяет совершить несколько запросов из-за того, что параметр вставляется в конце и с помощью точки с запятой можно указать конец первого запроса и сделать второй.

4. `python sqlmap.py -u "http://127.0.0.1:3000/fourth/?workerId=1" -p "workerId"`



Параметр workerId не подвержен инъекциям.

Выводы.

Протестировав доступ к БД с помощью ORM Sequelize на безопасность, выяснили, что два из четырех эндпоинта подвержены SQL инъекциям. Для противодействия таким инъекциям необходимо перед выполнением запросов форматировать ввод, если вводимая информация имеет определенную форму, то проверять на соблюдение формы, не предоставлять информацию об ошибках БД.

ПРИЛОЖЕНИЕ А

Pull request: https://github.com/moevm/sql-2023-1303/pull/62