Гр. 522403 Яцкевич Фёдор

Отчет на тему

**Тестирование на SQL инъекции**

Суть SQL инъекции состоит в том, что при изменении параметра SQL запроса на определенный текст, представляющий собой часть SQL запроса, результат выполнения запроса изменится.

Пример SQL инъекции:

Есть запрос поиска пользователя из базы по ID, принимающий ID в качестве параметра:

SELECT \*

FROM users

WHERE

id = ‘?’;

Ожидается, что параметр является целым числом. Но что если вместо этого параметр будет следующая строка?

1‘; DELETE FROM users;

В таком случае итоговый запрос будет следующий:

SELECT \*

FROM users

WHERE

id = ‘1‘; DELETE FROM users;’;

Вместо одного запроса было выполнено два. В лучшем случае результатом данного запроса будет ошибка. В худшем – сотрутся все данные из таблицы users.

Суть использования SQL инъекций состоит в том, что пользователь, пользующийся веб сервисом, взаимодействующим с базой данных, может изменить параметр http запроса или данные cookie, с определенной целью. Например, удаление данных веб сервиса, получение данных, недоступных простому пользователю.

Тестировать сервис на SQL инъекции на стороне клиента не имеет смысла, т.к. хакер может вручную заполнить параметр запроса или данные cookie SQL инъекцией (через консоль браузера).

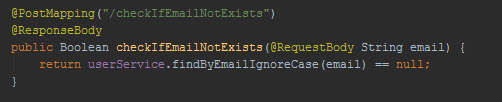
Таким образом приходим к тому что сервис тестировать необходимо с момента прихода запроса на сервер, т.к. тестовые данные нужно занести в параметр запроса.

Пример приведенный выше не совсем подходит для тестирования, т.к.

* он состоит из 2 запросов, что иногда автоматически может считаться системой как некорректное поведение, и второй запрос не проходит
* он выполняет операцию удаления, которая не пройдет в данной системе, т.к. таблица users привязана к таблице user\_role, и для удаления записи из users, необходимо сперва удалить соответствующие записи из user\_role.

Для примера тестирования SQL инъекций воспользуемся готовым сайтом на Java, Spring Boot, PostgreSQL. Будем тестировать метод контроллера

SecurityController.checkIfUsernameNotExists(String username)



Данный метод принимает в качестве параметра строку и возвращает булевое значение: **true, если пользователя не существует и false если он существует**.

Пусть параметр запроса – строка “user1” и пользователь с данным именем отсутствует в базе, при этом в базе есть пользователь с именем user.

Предположим, что данный сервис выполнит следующий запрос к базе:

SELECT \*

WHERE

username = ‘?’;

Теперь изменим параметр на следующую строку

user1’ OR username = ‘user

Результирующий запрос:

SELECT \*

WHERE

username = ‘user1’ OR username = ‘user’;

Т.е. получаем что, в случае если контроллер уязвим к SQL инъекциям, он вернет результат true даже если пользователь user1 существует.

Приступим к написанию тестов. Инициализируем базу данных и сконфигурируем тестовый класс перед началом тестов:

package fyodor.controller;  
  
import fyodor.model.Role;  
import fyodor.model.User;  
import fyodor.repository.RoleRepository;  
import fyodor.repository.UserRepository;  
import org.junit.Before;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.http.MediaType;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.test.context.junit4.SpringRunner;  
import org.springframework.test.web.servlet.MockMvc;  
import org.springframework.test.web.servlet.setup.MockMvcBuilders;  
import org.springframework.web.context.WebApplicationContext;  
  
import java.text.SimpleDateFormat;  
import java.util.Arrays;  
import java.util.HashSet;  
  
import static org.springframework.security.test.web.servlet.request.SecurityMockMvcRequestPostProcessors.*csrf*;  
import static org.springframework.security.test.web.servlet.setup.SecurityMockMvcConfigurers.*springSecurity*;  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*post*;  
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.*print*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*content*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.*RANDOM\_PORT*)  
public class SecurityControllerTestSQLInjections {  
  
 @Autowired private WebApplicationContext context;  
 @Autowired private UserRepository userRepository;  
 @Autowired private RoleRepository roleRepository;  
 @Autowired private PasswordEncoder passwordEncoder;  
  
 private MockMvc mvc;  
  
 @Before  
 public void setup() throws Exception {  
 mvc = MockMvcBuilders  
 .*webAppContextSetup*(context)  
 .apply(*springSecurity*())  
 .build();  
  
 initDB();  
 }  
  
 private void initDB() throws Exception {  
 userRepository.deleteAll();  
 roleRepository.deleteAll();  
 Role role = roleRepository.save(new Role(1L, "ROLE\_USER"));  
 userRepository.save(new User("user", "email1@mail.com", passwordEncoder.encode("password"),  
 new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SS").parse("2018-01-02 12:30:19.228"),  
 new HashSet<>(Arrays.*asList*(role))));  
 }.andExpect(*status*().is5xxServerError());  
 }  
}

Сперва напишем тест для корректных данных:

@Test  
public void checkIfUsernameNotExists\_userExists() throws Exception{  
 this.mvc.perform(*post*("/checkIfUsernameNotExists")  
 .param("username", "user")  
 .contentType(MediaType.*APPLICATION\_FORM\_URLENCODED*)  
 .with(*csrf*()))  
 .andDo(*print*())  
 .andExpect(*content*().string("false"));  
}

Данный тест был пройден успешно



Теперь напишем тест для SQL инъекции:

@Test  
public void checkIfUsernameNotExists\_sqlInjection() throws Exception{  
 this.mvc.perform(*post*("/checkIfUsernameNotExists")  
 .param("username", "user1' OR username='user")  
 .contentType(MediaType.*APPLICATION\_FORM\_URLENCODED*)  
 .with(*csrf*()))  
 .andDo(*print*())  
 .andExpect(*content*().string("true"));  
}



Данный тест был пройден успешно, и пользователь не был найден. В данном случае система приняла имя пользователя "user1' OR username='user" и попыталась найти его в БД. Пользователь не был найден, поэтому результат запроса – true, как и требовалось.

При провале SQL инъекции было возможно 2 вида поведения системы:

- возникновение ошибки, что привело бы к созданию исключения и его обработки

- распознание системой кавычек и корректная их обработка

В данном случае это привело к второму исходу, о чем свидетельствует выполнение условия

.andExpect(content().string(“true”))

Таким образом было произведено тестирование на SQL инъекцию.