Мукосов Станислав. Практическое задание: Анализ защищённости веб-приложений

1.Введение

Какие уязвимости будем искать? Опираясь на классификацию Top 10 Web Application Security Risks, существуют следующие виды:

- Инъекции (Injections).
- Нарушенная аутентификация (Broken Authentication).
- Раскрытие критически важных данных (Sensitive Data Exposure).
- Внешние объекты XML (XXE) (XML External Entities (XXE)).
- Нарушенный контроль доступа (Broken Access control).
- Неправильная конфигурация безопасности (Security misconfigurations).
- Межсайтовый скриптинг (XSS) (Cross Site Scripting (XSS)).
- Небезопасная десериализация (Insecure Deserialization).
- Использование компонентов с известными уязвимостями (Using Components with known vulnerabilities).
- Недостаточно подробные журналы и слабый мониторинг (Insufficient logging and monitoring).

Или как в иригинале:

A01:2021-Broken Access Control

A02:2021-Cryptographic Failures

A03:2021-Injection

A04:2021-Insecure

A05:2021-Security Misconfiguration

A06:2021-Vulnerable and Outdated Components

A07:2021-Identification and Authentication Failures

A08:2021-Software and Data Integrity Failures

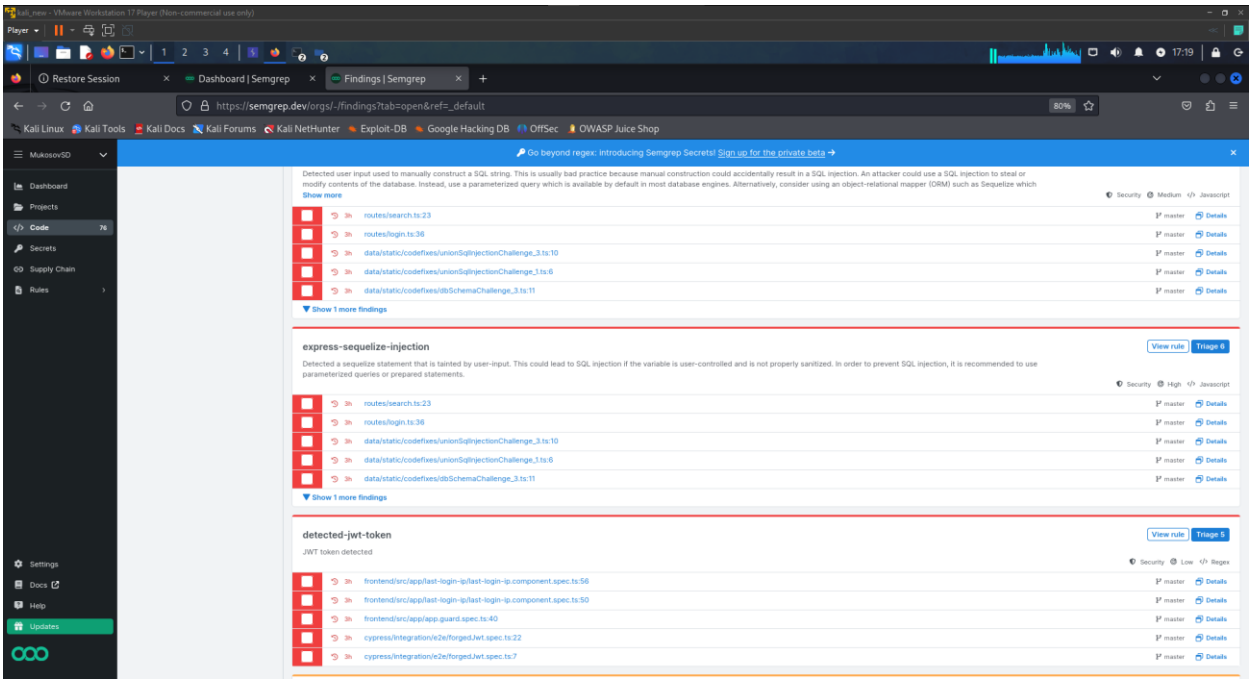
A09:2021-Security Logging and Monitoring Failures

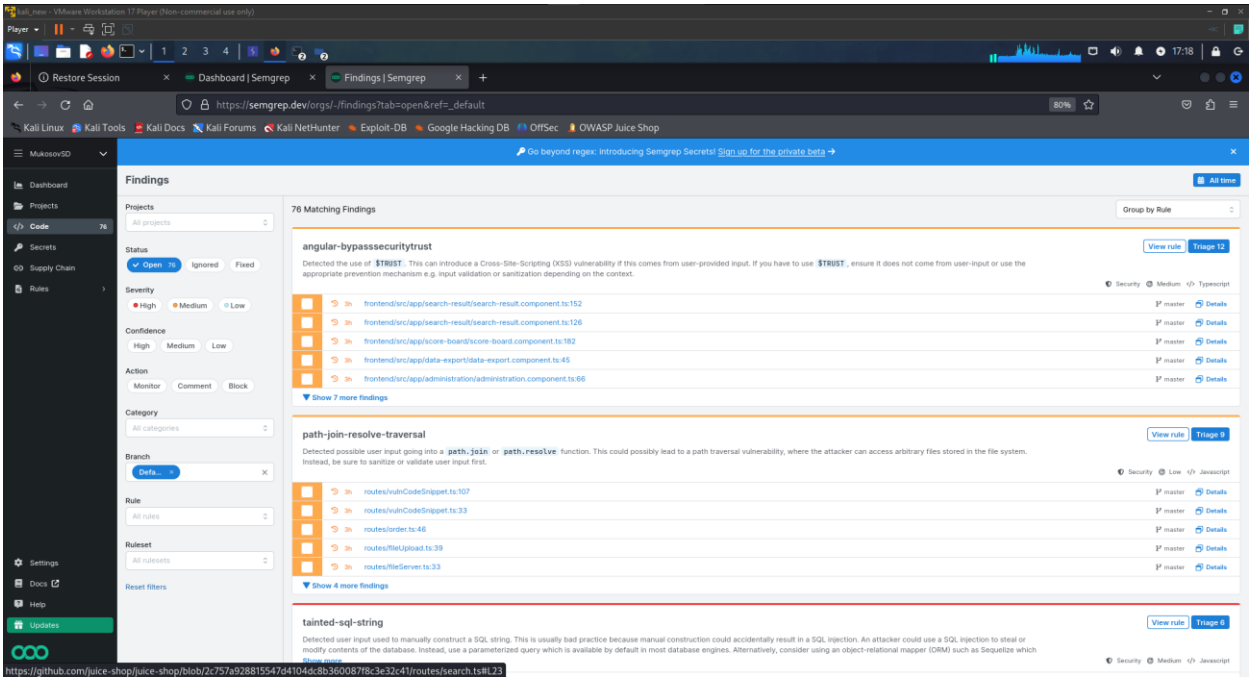
A10:2021-Server-Side Request Forgery

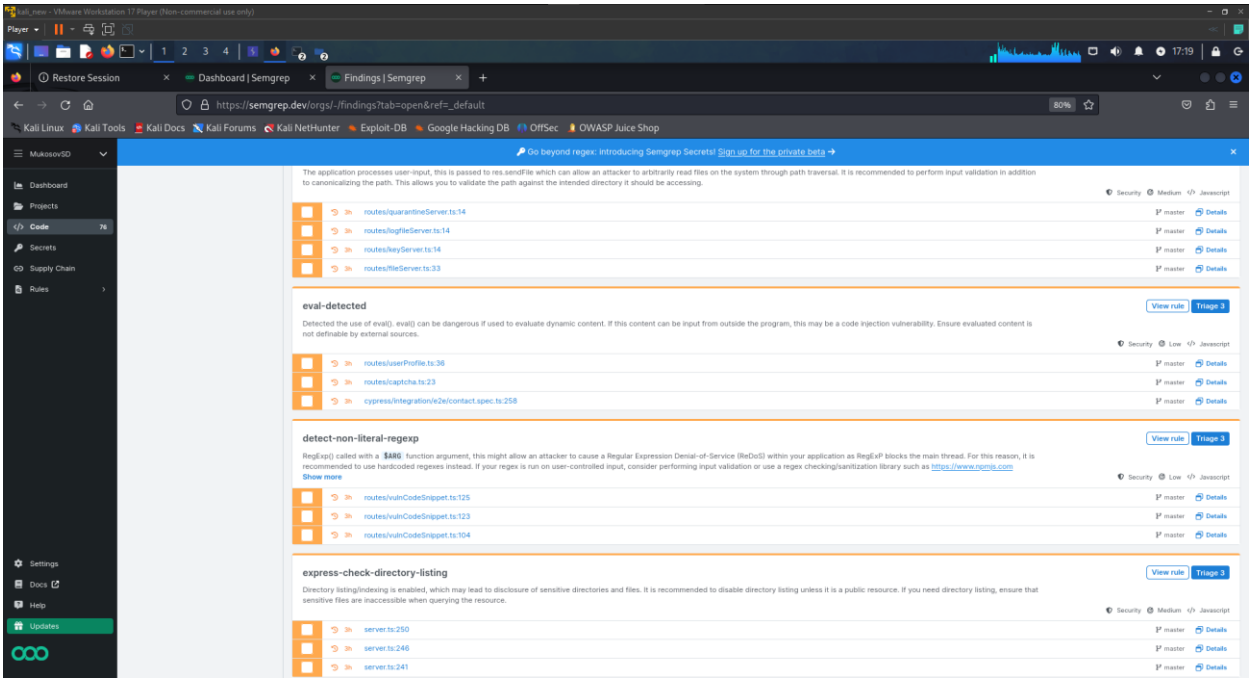
OWASP (Open Web Application Security Project) – это открытый некоммерческий фонд, который занимается вопросами обеспечения безопасности веб-приложений. Кроме крайне популярного OWASP Top 10, там разработали и постоянно развивают проект Juice Shop, призванный продемонстрировать самые часто встречаемые уязвимости в приложениях и «худшие практики» веб-разработки.

2. Результаты статического анализа

Знатно «попотев» с установкой **semgrep**, можно начинать работу. Просканируем веб-приложение juice shop и посмотрим результат:







Найдено **76** уязвимостей.

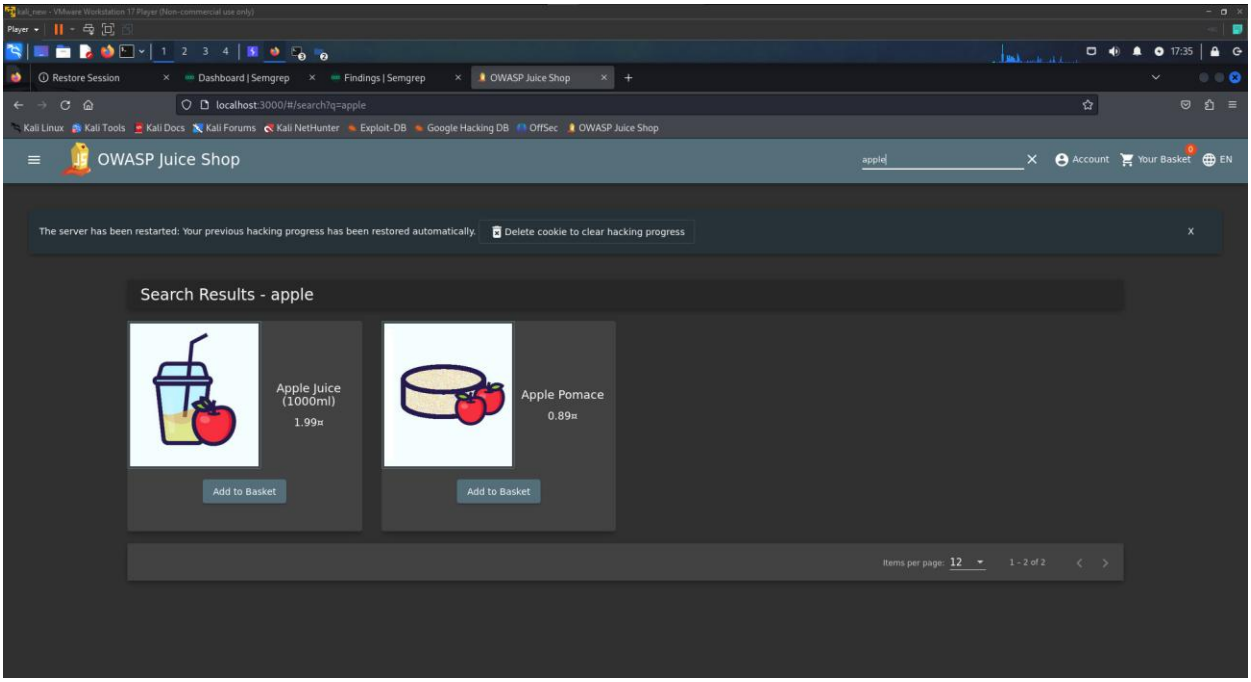
3. Уязвимости из OWASP Top-10, обнаруженные в результате статического анализа Juicy Shop.

- A01_2021-Broken_Access_Control (Forged Feedback).
- A01_2021-Broken_Access_Control (Forged Review).
- A01_2021-Broken_Access_Control (Manipulate Basket).
- A03:2021-Injection (Database Schema).
- A03:2021-Injection (Ephemeral Accountant).

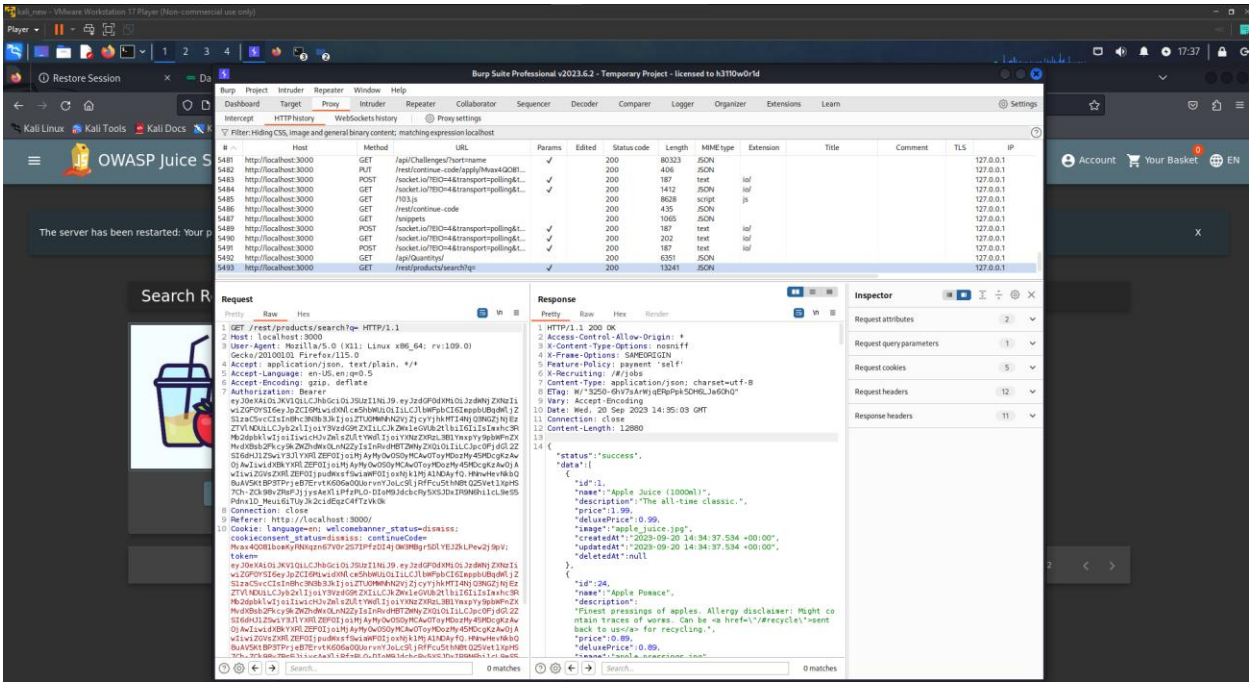
4. Демонстрация эксплуатации трёх уязвимостей из OWASP Top-10

Попробуем поэксплуатировать Injection (Database Schema)

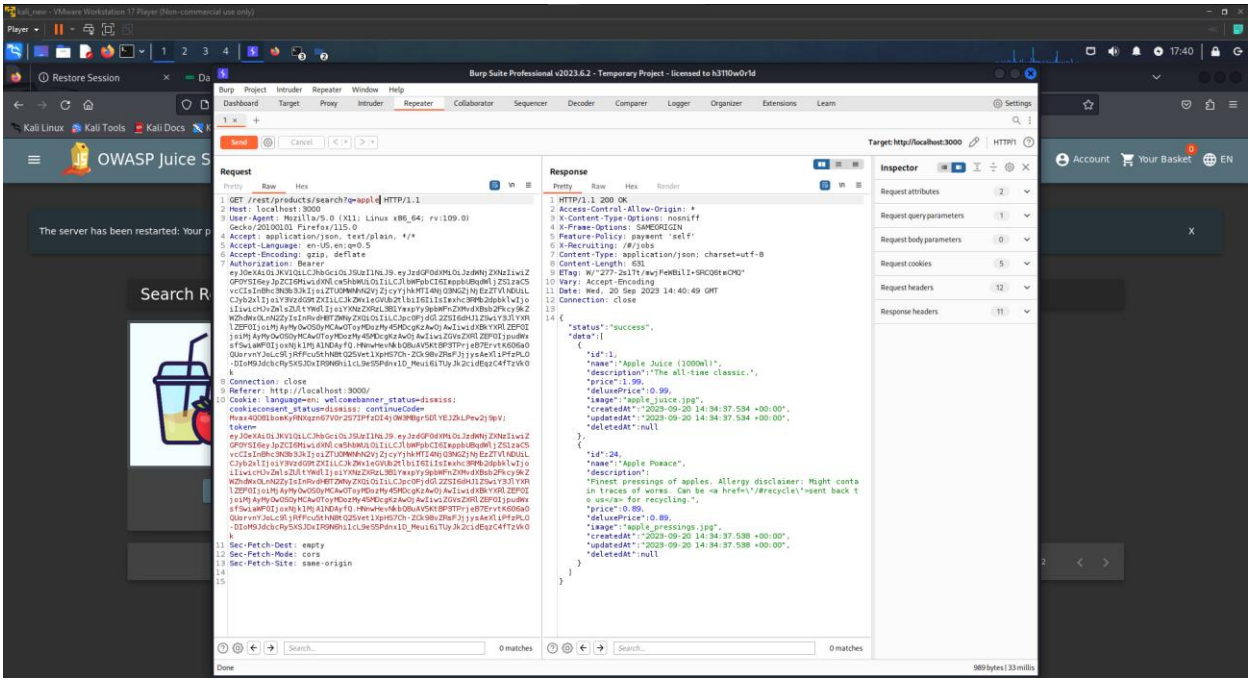
Вводим в окно поиска, например, apple и получаем результат



Запрос на поиск перехватываем Burp’ом и отправляем его в Repeater для модификации.

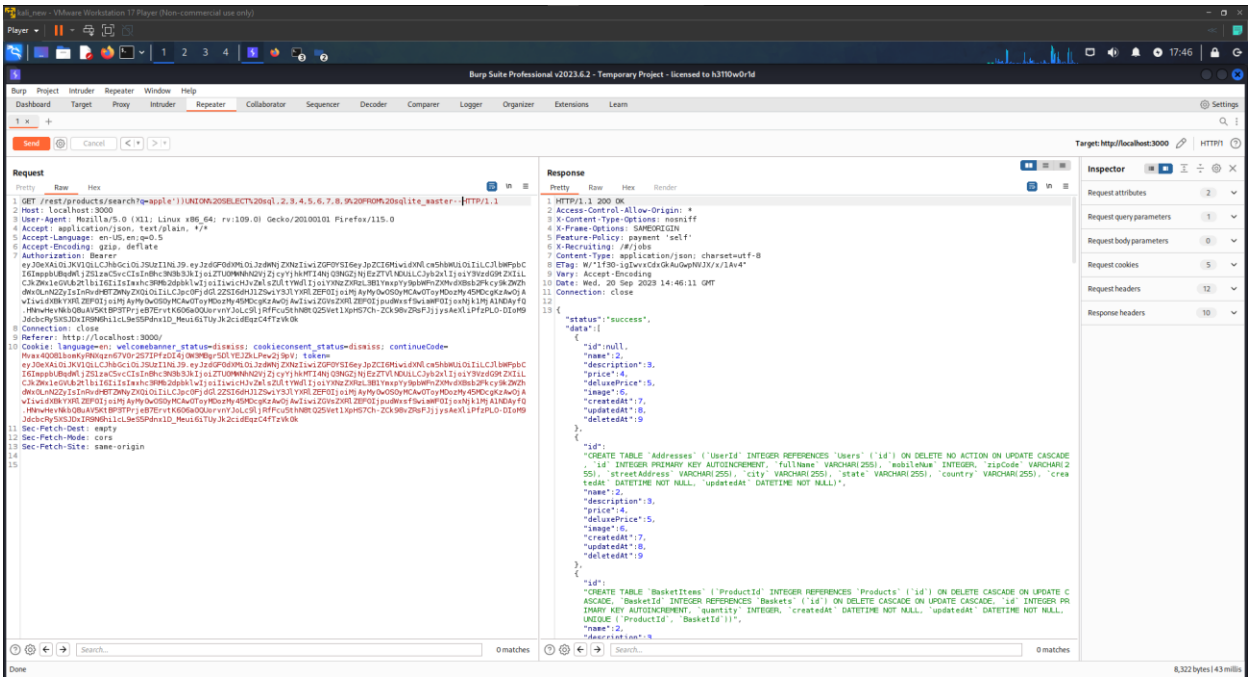


Подставляем снова значение apple и смотрим результат



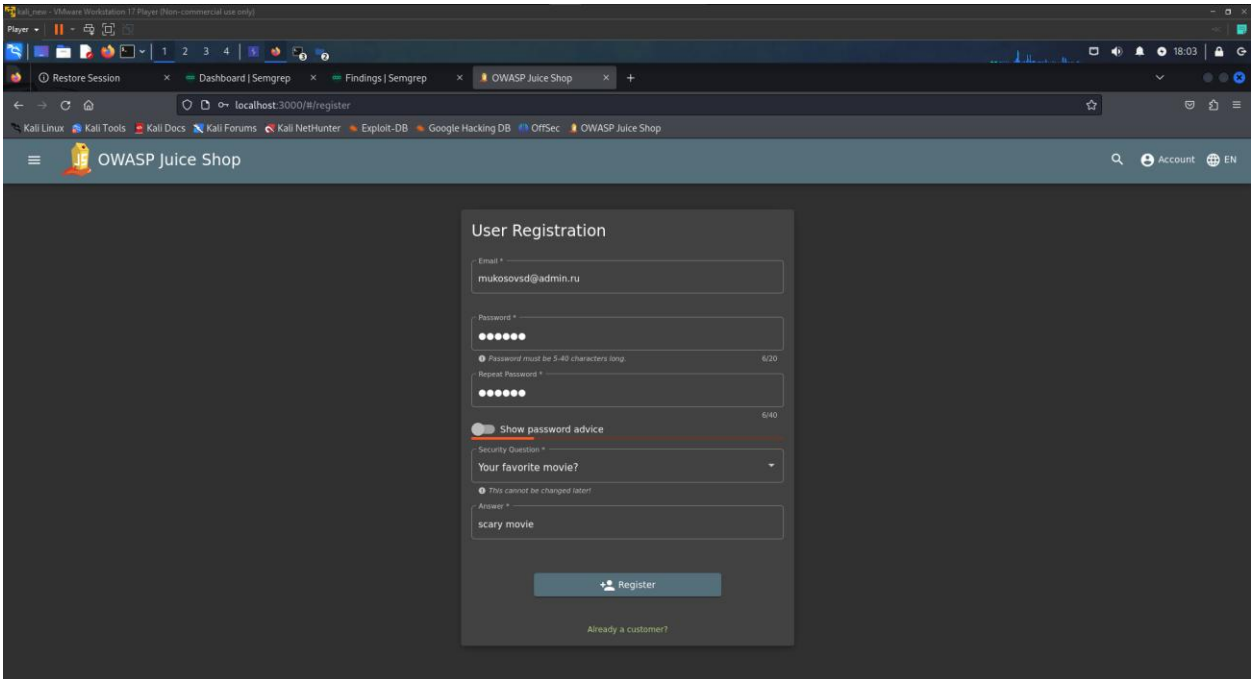
В результате “success”, а это значит можем попробовать подставить SQL-запрос. Пробуем подставить запрос найденный на дружественных git- репозиториях

apple'))UNION%20SELECT%20sql,2,3,4,5,6,7,8,9%20FROM%20sqlite_master—

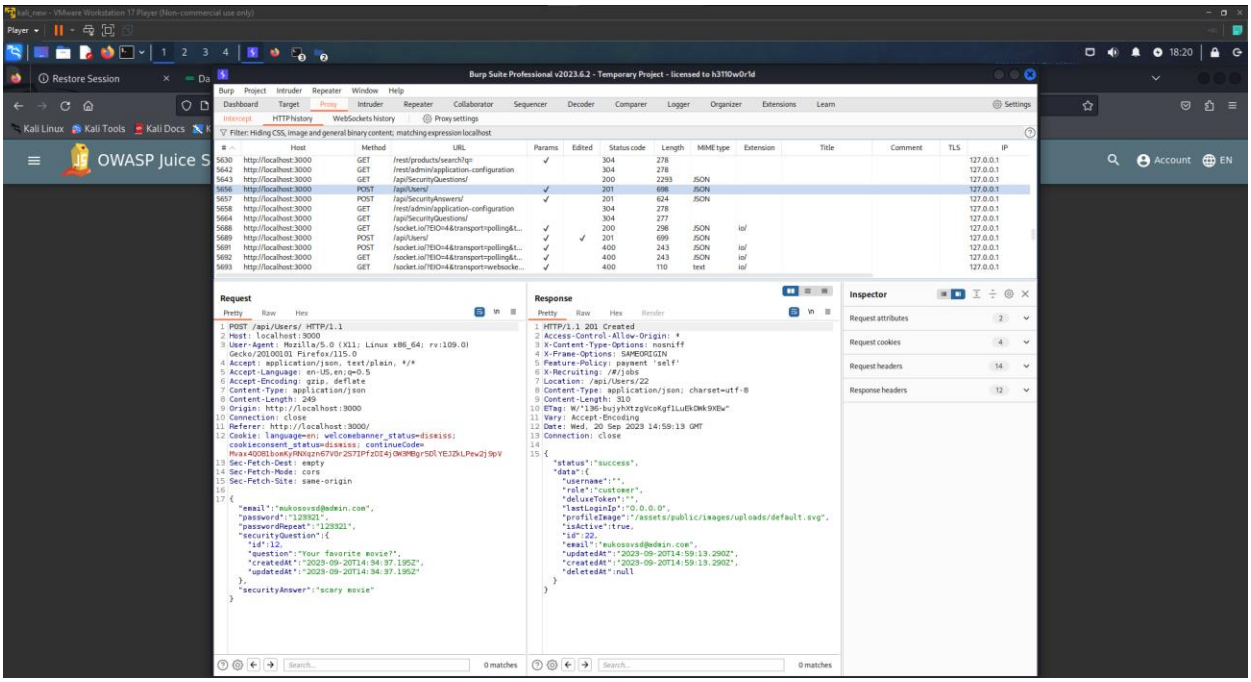


Получаем результат: схема таблиц базы доступна для просмотра.

Далее попробуем сделать Broken Access Control на примере Admin Registration
Регистрируем аккаунт на платформе, перехватываем и смотрим Burp’ом



Смотрим, какой ответ нам приходит



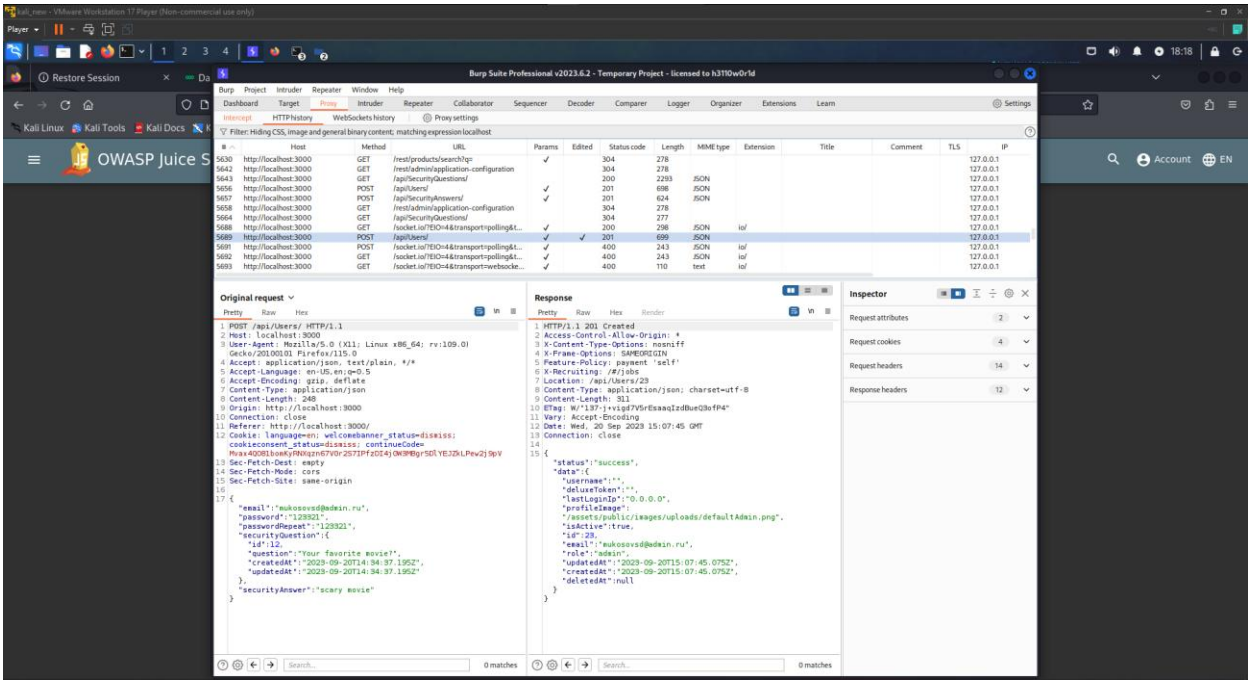
В ответе появился параметр “role” со значением “customer”
Попробуем зарегистрироваться еще раз и вручную подставить роль администратора.

Перехватываем наш запрос intercept’ом и подставляем значение.

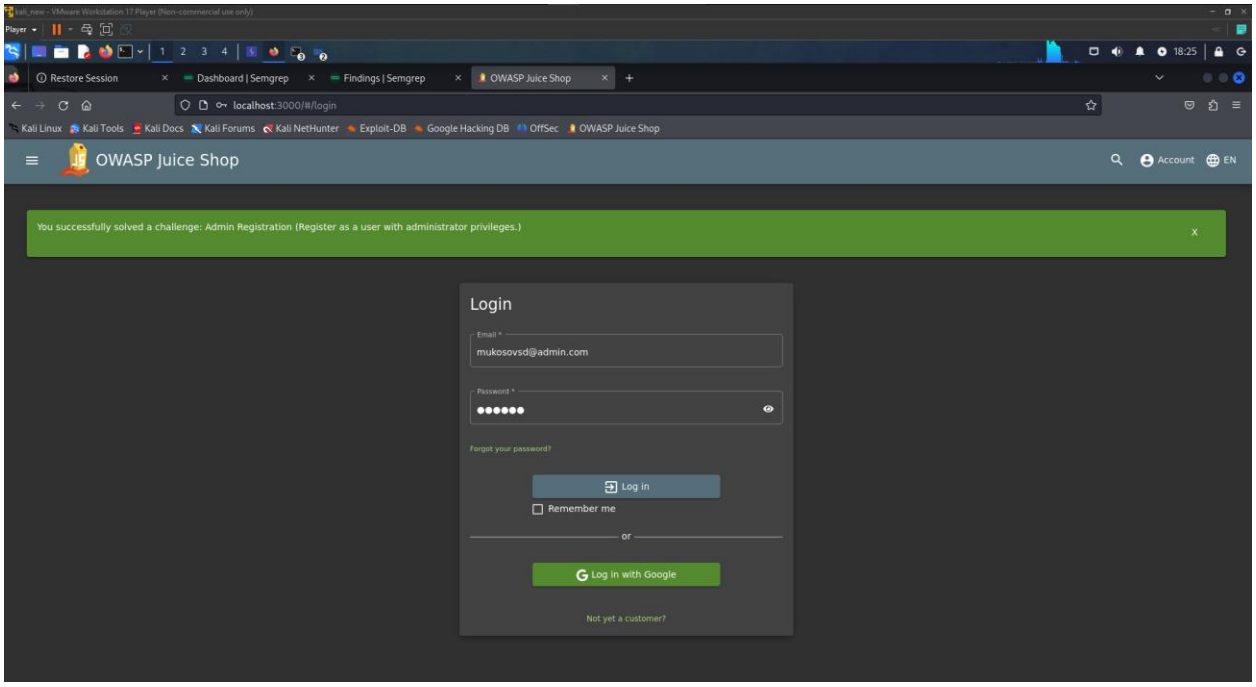
```
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 248
Origin: http://localhost:3000
Connection: close
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_stat
Mvax4QO8lbomKyRNXqzn67V0r2S7IPfzDI4j0W3MBgr5DlYEJZkLPew2j9pV
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"mukosovsd@admin.ru",
  "role":"admin",
  "password":"123321",
  "passwordRepeat":"123321",
  "securityQuestion":{
    "id":12,
    "question":"Your favorite movie?",
    "createdAt":"2023-09-20T14:34:37.195Z",
    "updatedAt":"2023-09-20T14:34:37.195Z"
  },
  "securityAnswer":"scary movie"
}
```

Проверяем полученный ответ от juice shop

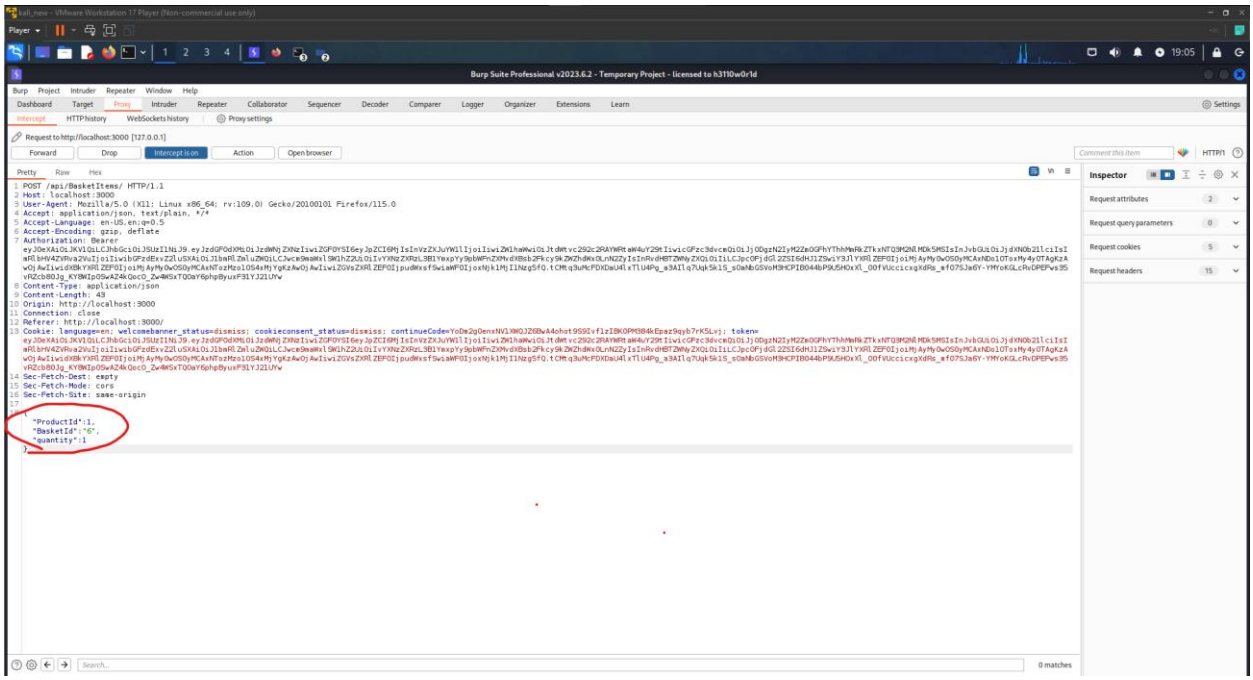


Успешно!

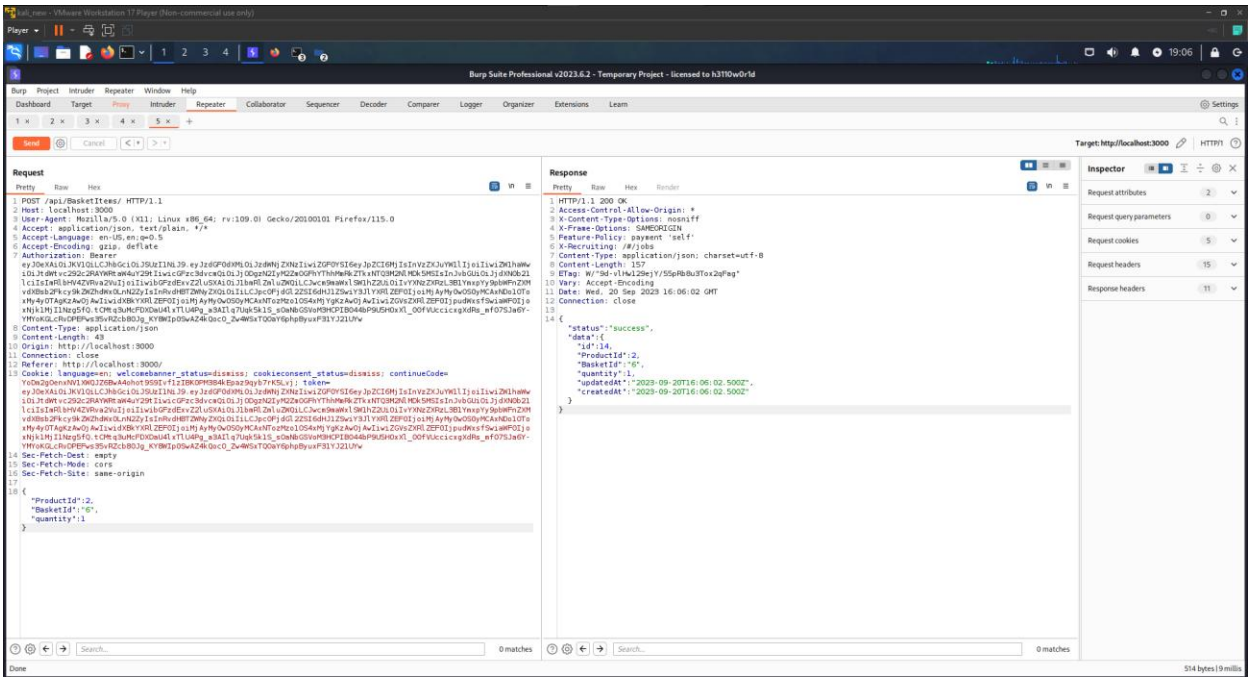


Далее попробуем поиграться с товарами. Например, положим товар другому пользователю в корзину))

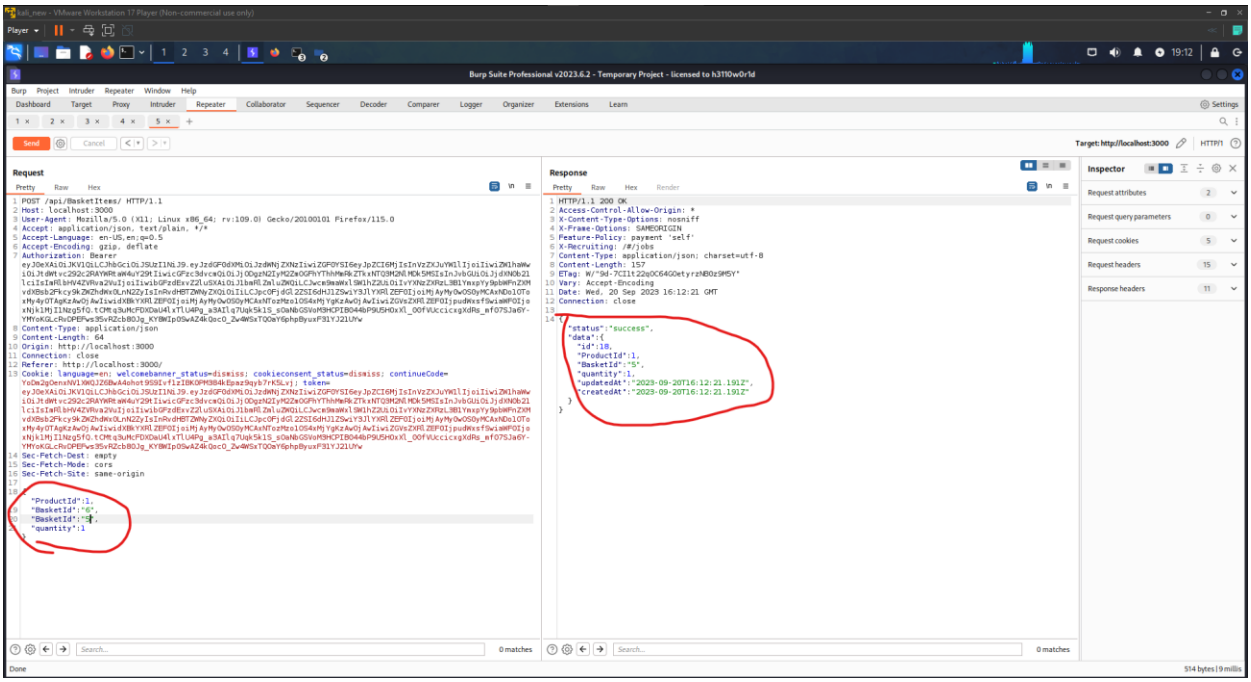
Кладем товар в свою корзину, чтобы перехватить запрос к веб-приложению



Отправляем этот запрос в repeator и отправляем juice shop

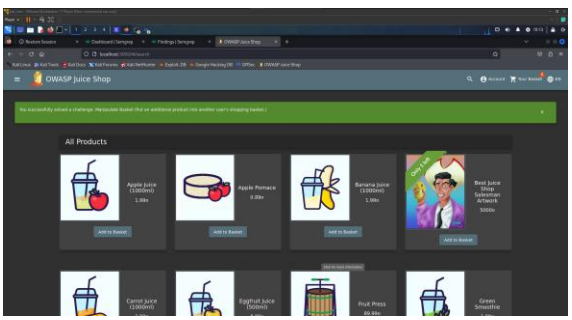


Видим, что у каждой корзины есть свой id. Попробуем сменить id и отправим запрос



В данном случае просто сменить id не получается, и приложение выдает ошибку.

НО! Если продублировать строку и во второй строке сменить id корзины, веб-приложение криминала не видит и успешно добавляет товар другому пользователю в корзину))



5. Рекомендации по устранению уязвимостей.

Broken Access Control.

Контроль доступа как мера эффективен в системе trusted server-side code or server-less API, где атакующий не способен модифицировать проверку access control check или metadata.
- Для всех ресурсов, кроме публичных: deny by default.

- Реализовать механизмы контроля доступа один раз на этапе разработки и использовать их в приложении, включающем минимизирование Cross-Origin Resource Sharing (CORS).
Implement access control mechanisms once and re-use them throughout the application, including

- Настройки контроля доступа должны установить разрешения на уровне объектов, не давая пользователю создавать, читать, обновлять или удалять любые записи.

- Уникальные настройки и требования организации должны быть установлены на уровне домена.

- Запретить web server directory listing и убедиться в недоступности метаданных для корневого каталога веб-сервера.

- Логгирование всех попыток неверного входа или использования учётных записей, уведомление администраторов при повторных срабатываниях.

- Установить API limit для минимизирования вреда от автоматизированых атак.

- Идентификаторы сессий должны быть обнулены на сервере после выхода из учётной записи. Stateless JWT tokens должны иметь короткий срок жизни.

- Разработчики и сотрудники QA обязаны тестировать функциональный контроль доступа и модели интеграции элементов.

Database Injection.

Считается, что для предотвращения данного типа атак необходимо использовать следующие два уровня обороны.

А. Параметризацию – где возможно, использовать структурные механизмы, которые обязывают разделятъ данные и команды. Этот механизм обеспечивается соответствующим кодированием строк, использованием кавычек, и т.д.

Б. Проверка ввода – значений для команд и относящихся к ним аргументов. Существуют разные подходы к проверки верности команд и их аргументов:
- При использовании команд, они сверяются со списком допустимых.
- Аргументы сверяются со списком позволенных и чётко определённых символов при вводе.
When it comes to the commands used, these must be validated against a list of allowed commands.
- Список разрешённых выражений использующих символы с заданной длинной.

- Необходимо убедиться, что метасимволы **& | ; $** > < \ \ **!`** и пробелы не являются частью RegularExpression. Например, следующее выражение позволяет только символы нижнего регистра и числа и не содержит метасимволы, длина ограничена 3-10 символами:
^[a-z0-9]{3,10}$.