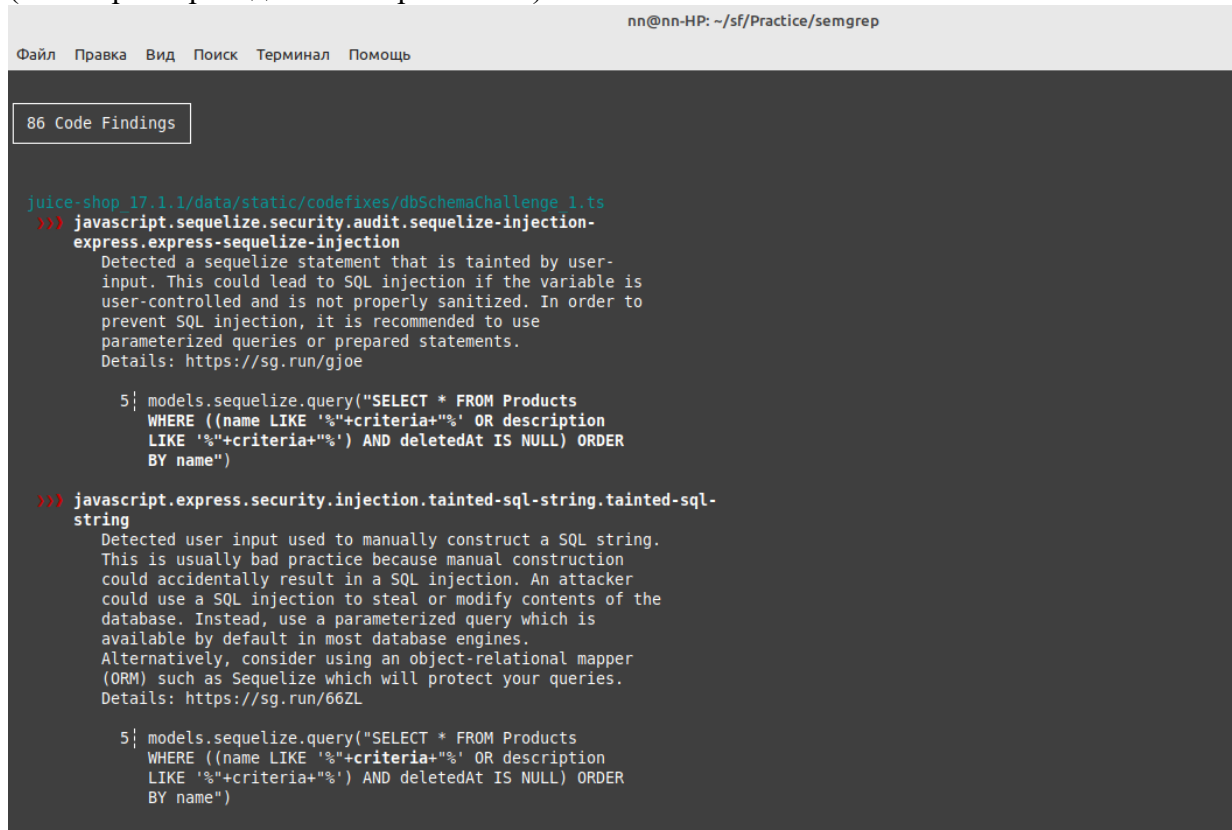


Практическое задание. Анализ защищённости веб-приложений

1. Введение.

OWASP Juice Shop — это преднамеренно уязвимое веб-приложение, разработанное в рамках проекта Open Web Application Security Project (OWASP) в качестве обучающей платформы для обеспечения безопасности веб-приложений. Оно предоставляет практикующим специалистам реалистичную и интерактивную среду, в которой они могут практиковать и улучшать свои навыки в выявлении и использовании уязвимостей веб-приложений.

2. Провел статический анализ Juice Shop при помощи Semgrep. Найдено 86 уязвимостей (некоторые приведены на скриншотах)



```
nn@nn-HP: ~/sf/Practice/semgrep

Файл  Правка  Вид  Поиск  Терминал  Помощь

86 Code Findings

juice-shop 17.1.1/data/static/codefixes/dbSchemaChallenge.1.ts
>>> javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection
    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.
    Details: https://sg.run/gjoe

    5; models.sequelize.query("SELECT * FROM Products
      WHERE ((name LIKE '%"criteria+%" OR description
      LIKE '%"criteria+%"') AND deletedAt IS NULL) ORDER
      BY name")

>>> javascript.express.security.injection.tainted-sql-string.tainted-sql-string
    Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries.
    Details: https://sg.run/66ZL

    5; models.sequelize.query("SELECT * FROM Products
      WHERE ((name LIKE '%"criteria+%" OR description
      LIKE '%"criteria+%"') AND deletedAt IS NULL) ORDER
      BY name")
```

juice-shop_17.1.1/data/static/codefixes/dbSchemaChallenge_3.ts

```
>>> javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection
```

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. Details: <https://sg.run/gjoe>

```
11| models.sequelize.query(`SELECT * FROM Products
    WHERE ((name LIKE '%${criteria}%' OR description
    LIKE '%${criteria}%') AND deletedAt IS NULL) ORDER
    BY name`)
```

```
>>> javascript.express.security.injection.tainted-sql-string.tainted-sql-string
```

Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries. Details: <https://sg.run/66ZL>

```
11| models.sequelize.query(`SELECT * FROM Products
    WHERE ((name LIKE '%${criteria}%' OR description
    LIKE '%${criteria}%') AND deletedAt IS NULL) ORDER
    BY name`)
```

juice-shop_17.1.1/data/static/codefixes/restfulXssChallenge_2.ts

```
) javascript.audit.detect-replaceall-sanitization.detect-replaceall-sanitization
```

Detected a call to `replaceAll()` in an attempt to HTML escape the string `tableData[i].description`. Manually sanitizing input through a manually built list can be circumvented in many situations, and it's better to use a well known sanitization library such as `sanitize-html` or `DOMPurify`. Details: <https://sg.run/AzoB>

```
59| tableData[i].description =
    tableData[i].description.replaceAll('<',
    '&lt;').replaceAll('>', '&gt;')
    |
```

```
) javascript.audit.detect-replaceall-sanitization.detect-replaceall-sanitization
```

Detected a call to `replaceAll()` in an attempt to HTML escape the string `tableData[i].description.replaceAll('<', '<')`. Manually sanitizing input through a manually built list can be circumvented in many situations, and it's better to use a well known sanitization library such as `sanitize-html` or `DOMPurify`. Details: <https://sg.run/AzoB>

```
59| tableData[i].description =
    tableData[i].description.replaceAll('<',
    '&lt;').replaceAll('>', '&gt;')
```

juice-shop_17.1.1/data/static/codefixes/unionSqlInjectionChallenge_1.ts

```
>>> javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection
```

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. Details: <https://sg.run/gjoe>

```
>>> javascript.express.security.injection.tainted-sql-string.tainted-sql-string
```

Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries. Details: <https://sg.run/66ZL>

```
6| models.sequelize.query(`SELECT * FROM Products
  WHERE ((name LIKE '%${criteria}%' OR description
    LIKE '%${criteria}%') AND deletedAt IS NULL) ORDER
    BY name`)
```

```
juice-shop_17.1.1/data/static/codexfixes/unionSqlInjectionChallenge_3.ts
```

```
>>> javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection
```

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. Details: <https://sg.run/gjoe>

```
10| models.sequelize.query(`SELECT * FROM Products
   WHERE ((name LIKE '%${criteria}%' OR description
     LIKE '%${criteria}%') AND deletedAt IS NULL) ORDER
     BY name`)
```

```
>>> javascript.express.security.injection.tainted-sql-string.tainted-sql-string
```

Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper

```
juice-shop_17.1.1/frontend/src/app/last-login-ip/last-login-ip.component.spec.ts
```

```
>>> generic.secrets.security.detected-jwt-token.detected-jwt-token
```

JWT token detected
Details: <https://sg.run/05N5>

```
50| localStorage.setItem('token', 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7Imxhc3Rmb2dpbkwiOiJoImS4YlJMuNCJ9fQ.RAKmdqwNypu0xv3SDjP04xMKvd1CddKvDFYDBfUt3bg')
   |
56| localStorage.setItem('token', 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7fX0.bVBhVll6IaeR3aUdo0eyR8YZe2S2DfhGAxTGfd9enLw')
```

```
juice-shop_17.1.1/frontend/src/app/search-result/search-result.component.ts
```

```
>> typescript.angular.angular-route-bypass-security-trust.angular-route-bypass-security-trust
```

Untrusted input could be used to tamper with a web page rendering, which can lead to a Cross-site scripting (XSS) vulnerability. XSS vulnerabilities occur when untrusted input executes malicious JavaScript code, leading to issues such as account compromise and sensitive information leakage. Validate the user input, perform contextual output encoding, or sanitize the input. A popular library used to prevent XSS is DOMPurify. You can also use libraries and frameworks such as Angular, Vue, and React, which offer secure defaults when rendering input. Details: <https://sg.run/JpBW>

```
151| this.searchValue =
    this.sanitizer.bypassSecurityTrustHtml(queryParam
    ) // vuln-code-snippet vuln-line localXssChallenge
    xssBonusChallenge
```

```
juice-shop_17.1.1/frontend/src/hacking-instructor/helpers/helpers.ts
```

```
>> javascript.lang.security.audit.prototype-pollution.prototype-pollution-loop.prototype-pollution-loop
```

```

juice-shop_17.1.1/routes/b2bOrder.ts
>> javascript.express.security.audit.express-detect-notevil-usage.express-
detect-notevil-usage
  Detected usage of the `notevil` package, which is
  unmaintained and has vulnerabilities. Using any sort of
  `eval()` functionality can be very dangerous, but if you
  must, the `eval` package is an up to date alternative. Be
  sure that only trusted input reaches an `eval()` function.
  Details: https://sg.run/W70E

22| vm.runInContext('safeEval(orderLinesData)',
   | sandbox, { timeout: 2000 })

juice-shop_17.1.1/routes/captcha.ts
>> javascript.browser.security.eval-detected.eval-detected
  Detected the use of eval(). eval() can be dangerous if used
  to evaluate dynamic content. If this content can be input
  from outside the program, this may be a code injection
  vulnerability. Ensure evaluated content is not definable by
  external sources.
  Details: https://sg.run/7ope

23| const answer = eval(expression).toString() //
   | eslint-disable-line no-eval

juice-shop_17.1.1/routes/chatbot.ts
>> javascript.express.security.injection.raw-html-format.raw-html-format
  User data flows into the host portion of this manually-
  constructed HTML. This can introduce a Cross-Site-Scripting
  (XSS) vulnerability if this comes from user-provided input.
  Consider using a sanitization library such as DOMPurify to
  sanitize the HTML within.
  Details: https://sg.run/5D03

198| body: bot.training.state ?
    | bot.greet(`${user.id}`) :
    | `${config.get<string>('application.chatBot.name')}`
    | isn't ready at the moment, please wait while I set
    | things up`

juice-shop_17.1.1/routes/dataErasure.ts

```

3. Уязвимости в веб-приложении Juice Shop из OWASP TOP 10

A03:2021-Injection - Database Schema

A03:2021-Injection - Login Admin

A03:2021-Injection - Ephemeral Accountant

A01_2021-Broken_Access_Control - Manipulate Basket

A01_2021-Broken_Access_Control - View Basket

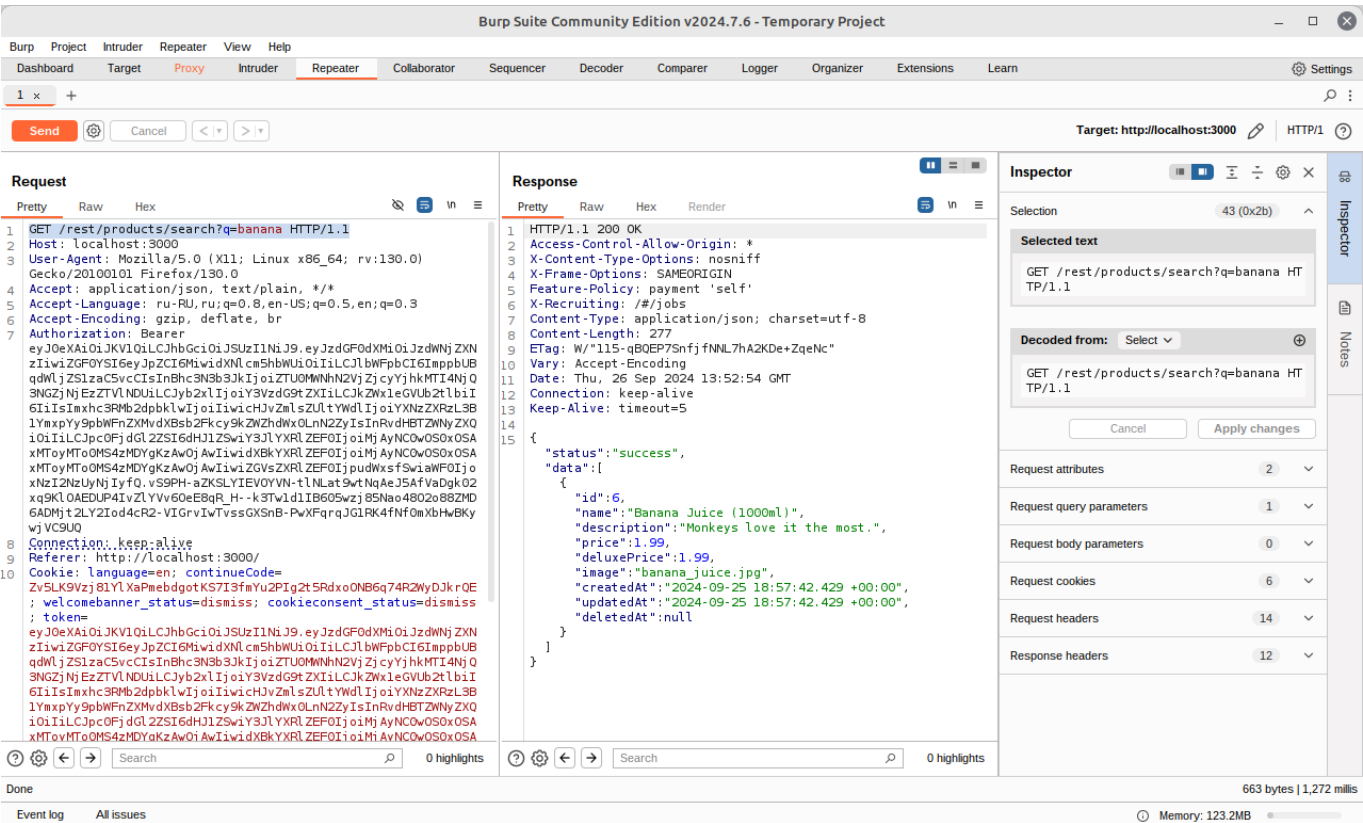
A03:2021-Injection - CWE-79: Cross-site Scripting, CWE-89: SQL Injection, and CWE-73: External Control of File Name or Path

A01_2021-Broken_Access_Control - CWE-200: Exposure of Sensitive Information to an Unauthorized Actor, CWE-201: Insertion of Sensitive Information Into Sent Data, and CWE-352: Cross-Site Request Forgery.

A03:2021-Injection - Database Schema

[illegible]

Проверил доходят ли запросы до сервера, добавив в строку наименование товара "banana" и отправил через репитер



Введя далее " " и вновь отправив запрос получил ошибку SQLITE_ERROR, из которой стало понятно, что имею дело с базой данных sqlite. Из дополнительных ресурсов по sqlite стало понятно, что главную таблицу sqlite можно запросить, выбрать нужный параметр sql и получить всю схему. Пример запроса:

```
SELECT sql
FROM sqlite_master
```

[illegible]

The image displays two side-by-side screenshots of a code editor interface, likely VS Code, showing the output of a database query. Both screenshots have a dark theme and a sidebar on the left with icons for Explorer, Search, and Run and Debug.

The left screenshot has a tab labeled "Response" and shows a JSON array of objects. The first object is a product with fields like `mobileNum`, `zipCode`, `streetAddress`, `city`, `state`, `country`, `createdAt`, `updatedAt`, and `deletedAt`. The second object is a `BasketItem` with fields like `ProductId`, `BasketId`, `quantity`, `createdAt`, `updatedAt`, and `deletedAt`.

The right screenshot also has a tab labeled "Response" and shows a JSON array of objects. The first object is a `Basket` with fields like `id`, `name`, `description`, `price`, `deluxePrice`, `image`, `createdAt`, `updatedAt`, and `deletedAt`. The second object is a `Captchas` entry with fields like `id`, `captchaId`, `captcha`, `answer`, `createdAt`, `updatedAt`, and `deletedAt`.

The screenshot displays a web browser window with the OWASP Juice Shop interface. The top section shows a 'Response' tab with a JSON object containing two database schema definitions. The first schema is for a 'Users' table, and the second is for a 'Wallets' table. The bottom section shows the OWASP Juice Shop homepage with a success message and challenge progress.

```
{
  "id": 1,
  "name": "Users",
  "description": "Users",
  "price": 4,
  "deluxePrice": 5,
  "image": "https://assets.owasp-juice.shop/assets/images/uploads/default.svg",
  "createdAt": "2020-01-01T00:00:00.000Z",
  "updatedAt": "2020-01-01T00:00:00.000Z",
  "deletedAt": null
}, {
  "id": 2,
  "name": "Wallets",
  "description": "Wallets",
  "price": 4,
  "deluxePrice": 5,
  "image": "https://assets.owasp-juice.shop/assets/images/uploads/default.svg",
  "createdAt": "2020-01-01T00:00:00.000Z",
  "updatedAt": "2020-01-01T00:00:00.000Z",
  "deletedAt": null
}
```

The OWASP Juice Shop interface shows a success message: "You successfully solved a challenge: Database Schema (Exfiltrate the entire DB schema definition via SQL Injection.)". The progress bar indicates 6% completion for Hacking Challenges, 0% for Coding Challenges, and 6/169 Challenges Solved.

До этого использовал различные варианты запроса и корректировал, опираясь на пояснения к ошибкам.

В итоговом запросе:

Union - объединяет запросы к БД

FROM sqlite_master - указывает на получение информации обо всех объектах БД

2,3,4,5,6,7,8,9 - так как в таблице есть определенное количество столбцов, но которое мне неизвестно, то путем подбора и опираясь на вывод запроса, стало в итоге понятно, что их 9. (на меньшее количество выводилась ошибка запроса).

Получение доступа администратора через sql инъекцию.

Burp Suite Community Edition v2024.7.6 - Temporary Project																		
Burp	Project	Intruder	Repeater	View	Help													
Dashboard	Target	Proxy	Intruder	Repeater	Collaborator	Sequencer	Decoder	Comparer	Logger	Organizer	Extensions	Learn	Settings					
Intercept	HTTP history	WebSockets history		Proxy settings														
Filter settings: Hiding CSS, image and general binary content																		
#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time			
451	http://localhost:3000	GET	/socket.io/?EIO=4&transport=polling...	✓					io/				127.0.0.1		18:20:56.20			
452	http://localhost:3000	GET	/rest/admin/application-configuration										127.0.0.1		18:20:59.20			
453	http://localhost:3000	GET	/rest/user/whoami										127.0.0.1		18:21:10.20			
454	http://localhost:3000	GET	/rest/user/whoami										127.0.0.1		18:21:10.20			
455	http://localhost:3000	POST	/rest/user/login	✓									127.0.0.1		18:21:10.20			
456	http://localhost:3000	GET	/socket.io/?EIO=4&transport=polling...	✓					io/				127.0.0.1		18:21:18.20			
457	https://snjdvw73vnmhs.statu...	GET	/api/v2/status.json					script	json			✓	108.157.229.101		18:21:22.20			
458	http://localhost:3000	GET	/socket.io/?EIO=4&transport=polling...	✓					io/				127.0.0.1		18:21:43.20			
459	http://localhost:3000	GET	/socket.io/?EIO=4&transport=polling...	✓					io/				127.0.0.1		18:22:08.20			
460	http://localhost:3000	GET	/										127.0.0.1		18:22:09.20			
461	https://snjdvw73vnmhs.statu...	GET	/api/v2/status.json					script	json			✓	108.157.229.40		18:22:22.20			

Burp Suite Community Edition v2024.7.6 - Temporary Project

Burp Project Intruder Repeater View Help

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings

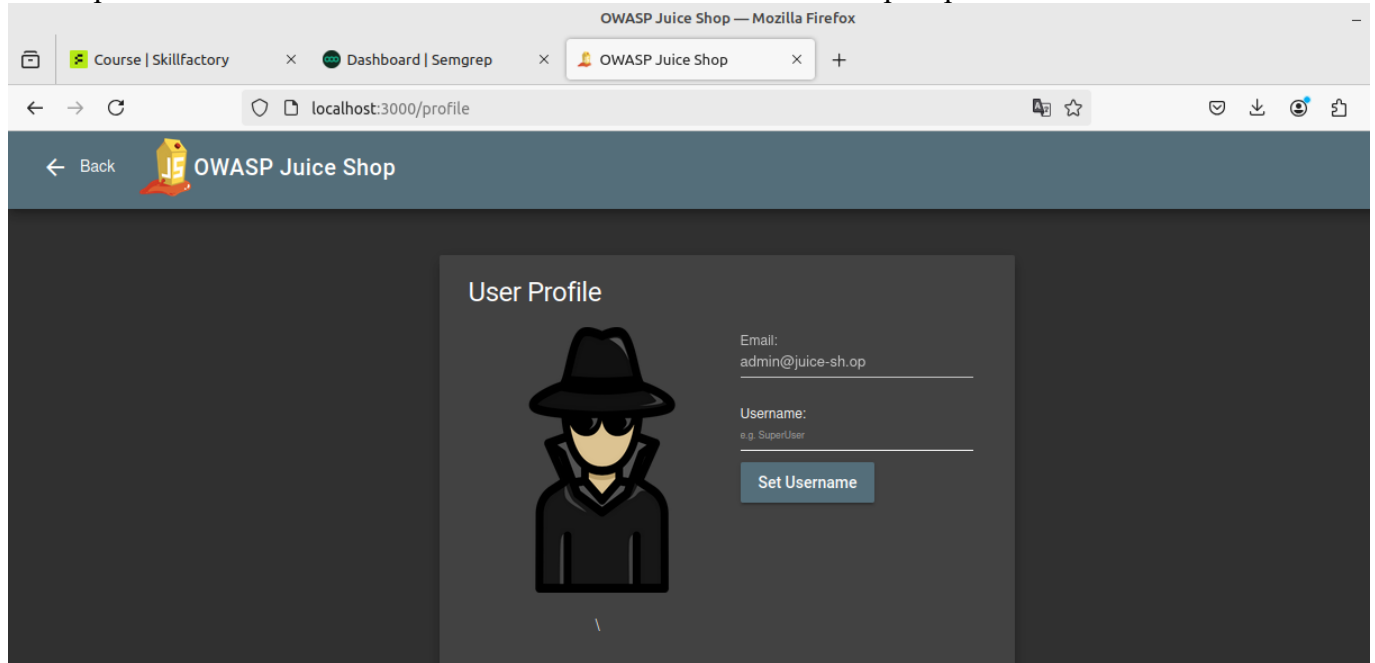
1 x 2 x +

Send Cancel < >

Target: http://localhost:3000 HTTP/1

Request		Response		Inspector
Pretty	Raw	Pretty	Raw	
<pre>1 POST /rest/user/login HTTP/1.1 2 Host: localhost:3000 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0 4 Accept: application/json, text/plain, /* 5 Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3 6 Accept-Encoding: gzip, deflate, br 7 Content-Type: application/json 8 Content-Length: 47 9 Origin: http://localhost:3000 10 Connection: keep-alive</pre>		<pre>2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 799 9 ETag: W/"31f-98pMQ5CTLswcO9lkMKnrpSKd4mg" 10 Vary: Accept-Encoding 11 Date: Thu, 26 Sep 2024 15:33:18 GMT 12 Connection: keep-alive</pre>		<ul style="list-style-type: none"> Request attributes 2 Request query parameters 0 Request cookies 5 Request headers 15 Response headers 12

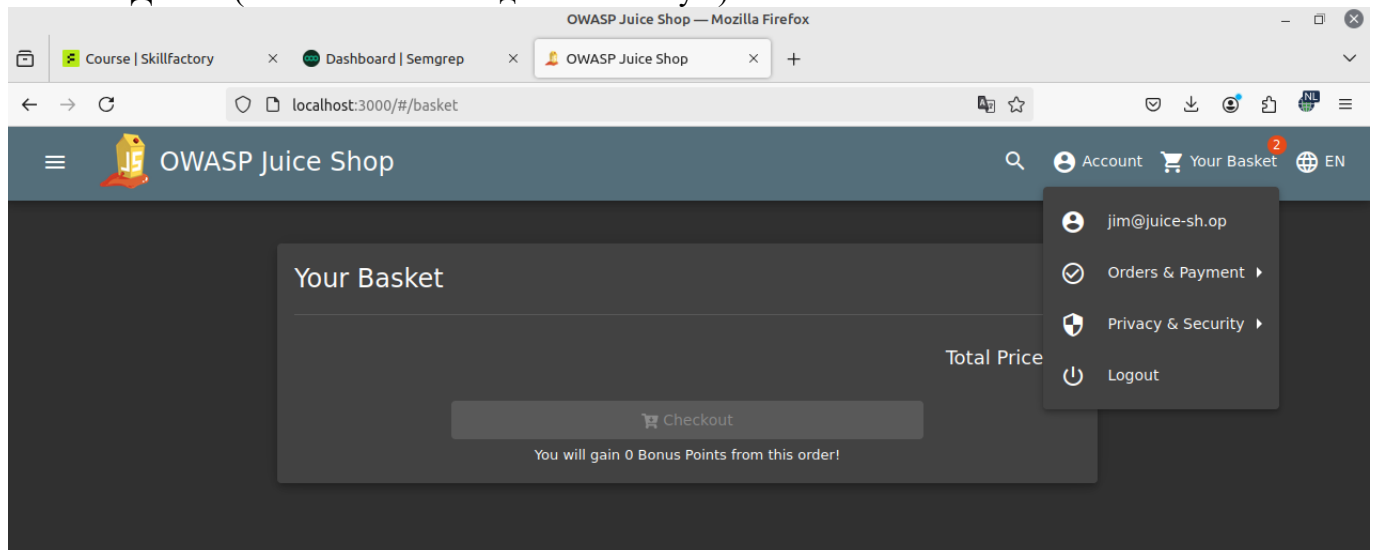
Произошел успешный вход под администратором. Под администратором удалось войти, потому что первым пользователем в базе данных был именно администратор.



A01_2021-Broken_Access_Control - View Basket

Просмотр корзины другого пользователя.

Для реализации запустил burp suite, зашел в корзину на Juice shop под известной мне уже учётной записью Джима (можно войти и создав свою новую).



Открыл burp suite и нашел запрос, в котором видно, что это корзина под номером 2.

Burp Suite Community Edition v2024.7.6 - Temporary Project

Burp Project Intruder Repeater View Help

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings

Intercept HTTP history WebSockets history Proxy settings

Filter settings: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time
512	https://www.google.com	GET	/complete/search?client=firefox&c...									✓	142.250.102.104		19:00:07.2f
513	http://localhost:3000	GET	/rest/basket/2										127.0.0.1		19:00:23.2f
514	http://localhost:3000	GET	/rest/user/whoami										127.0.0.1		19:00:23.2f
515	https://snjdw73nmhs.statu...	GET	/api/v2/status.json					script	json			✓	108.157.229.101		19:00:24.2f
516	https://snjdw73nmhs.statu...	GET	/api/v2/status.json					script	json			✓	108.157.229.101		19:01:24.2f

Request

Pretty Raw Hex

```
1 GET /rest/basket/2 HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJkdWlnZmZlbnRlLnNpdG90YSI6eyJpZCI6MiwiZW5kcm5kbWUiOiIiLCJlbWFPbCI6ImppbUBqdWljZSIzaC5vcCIsInBhc3N3B3JkIjoia2tUOMWNHnN2vjZcyYjhkMTI4NjQ3NGZjNjEzTVNDUuLjYybzxlIjoyY3Vzdg9tZXIiLCJkZWxleGVub2t1biI6IiIsImxhc3Rmb2dpbkklIjoydWskZWpibmkIiwicHVjZmlsZSUtYWdlIjoieYXNZXRzLR3BlYmpyY9pbWFnZXMydXBsb2Fkcy9kZWZhdmw0LnN2ZyIsInRvdHBTZWNyZXQlOiIiLCJpc0FjdGJgZSI6dHJlZSwiY3JlYXRRLFEFOFIjoimAynNCOWOSoyNSAxOdo1Nzo0S43NjUgKzAwQAwIiwidXBkYXRLLFEFOFIjoimAynNCOWOSoyNSAxNtoXnozyHy44OTUgKzAwQAwIiwidGVzZWRLLFEFOFIjpuDwxsfswiawFOFIjoXnzI3MzY2MzU0f0.NZEIHogugZPaB6_OCW_vBCDUgePCBNmzmktLTlHlmlqx9tmeuDQTObHgN8h_2gYCM4HhdUzu ralQyNs6GQUXeSSknT_54rBEvihPT_9V93HrInjN-7cIKRVtCSRtuh_huoj rHVugff4I4xXOAUVUQKHixqbQZtAewPXucys
8 Connection: keep-alive
9 Referer: http://localhost:3000/
0 Cookie: language=en; continueCode=ZySLK9Vzj8LYlXaPwebgdKS7I3fmYu2PIgt2SRdxoONB6q74R2WyDJkrQE; welcomebanner_status=dismiss; cookieconsent_status=dismiss; code-fixes-component-format=LineByLine; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJkdWlnZmZlbnRlLnNpdG90YSI6eyJpZCI6MiwiZW5kcm5kbWUiOiIiLCJlbWFPbCI6ImppbUBqdWljZSIzaC5vcCIsInBhc3N3B3JkIjoia2tUOMWNHnN2vjZcyYjhkMTI4NjQ3NGZjNjEzTVNDUuLjYybzxlIjoyY3Vzdg9tZXIiLCJkZWxleGVub2t1biI6IiIsImxhc3Rmb2dpbkklIjoydWskZWpibmkIiwicHVjZmlsZSUtYWdlIjoieYXNZXRzLR3BlYmpyY9pbWFnZXMydXBsb2Fkcy9kZWZhdmw0LnN2ZyIsInRvdHBTZWNyZXQlOiIiLCJpc0FjdGJgZSI6dHJlZSwiY3JlYXRRLFEFOFIjoimAynNCOWOSoyNSAxOdo1Nzo0S43NjUgKzAwQAwIiwidXBkYXRLLFEFOFIjoimAynNCOWOSoyNSAxNtoXnozyHy44OTUgKzAwQAwIiwidGVzZWRLLFEFOFIjpuDwxsfswiawFOFIjoXnzI3MzY2MzU0f0.NZEIHogugZPaB6_OCW_vBCDUgePCBNmzmktLTlHlmlqx9tmeuDQTObHgN8h_2gYCM4HhdUzu ralQyNs6GQUXeSSknT_54rBEvihPT_9V93HrInjN-7cIKRVtCSRtuh_huoj rHVugff4I4xXOAUVUQKHixqbQZtAewPXucys
```

Inspector

Request attributes 2

Request cookies 6

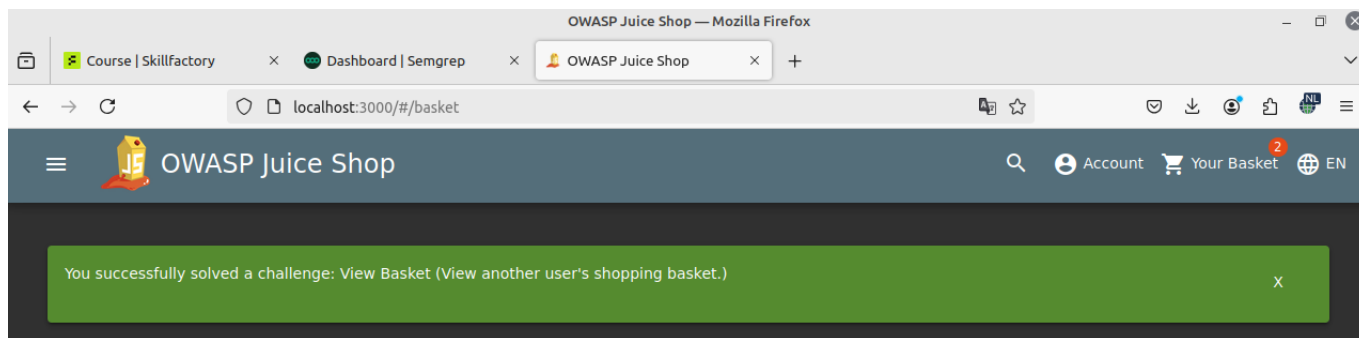
Request headers 14

Event log (2) All issues 0 highlights

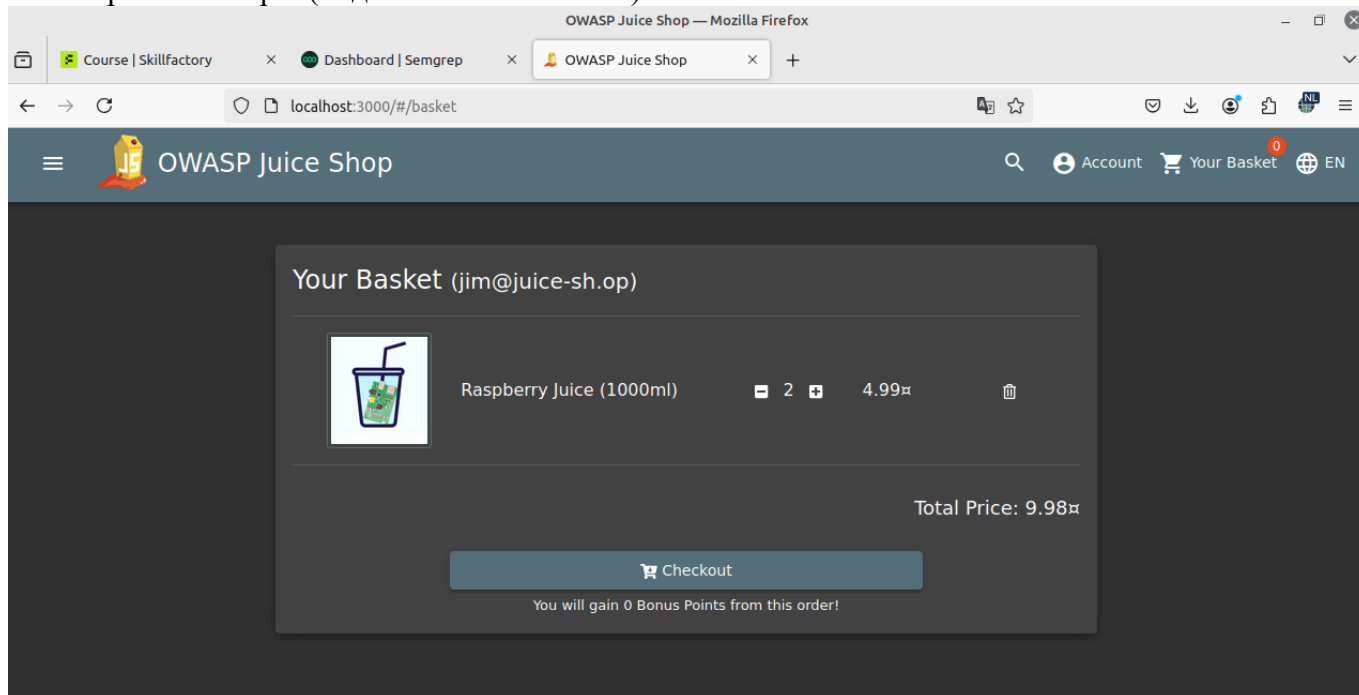
Memory: 129.6MB

Через репитер отправил запрос, изменив корзину на номер 1, и смог просмотреть состав у другого пользователя (3 наименования - Apple Juice (1000ml), Orange Juice (1000ml), Eggfruit Juice (500ml))

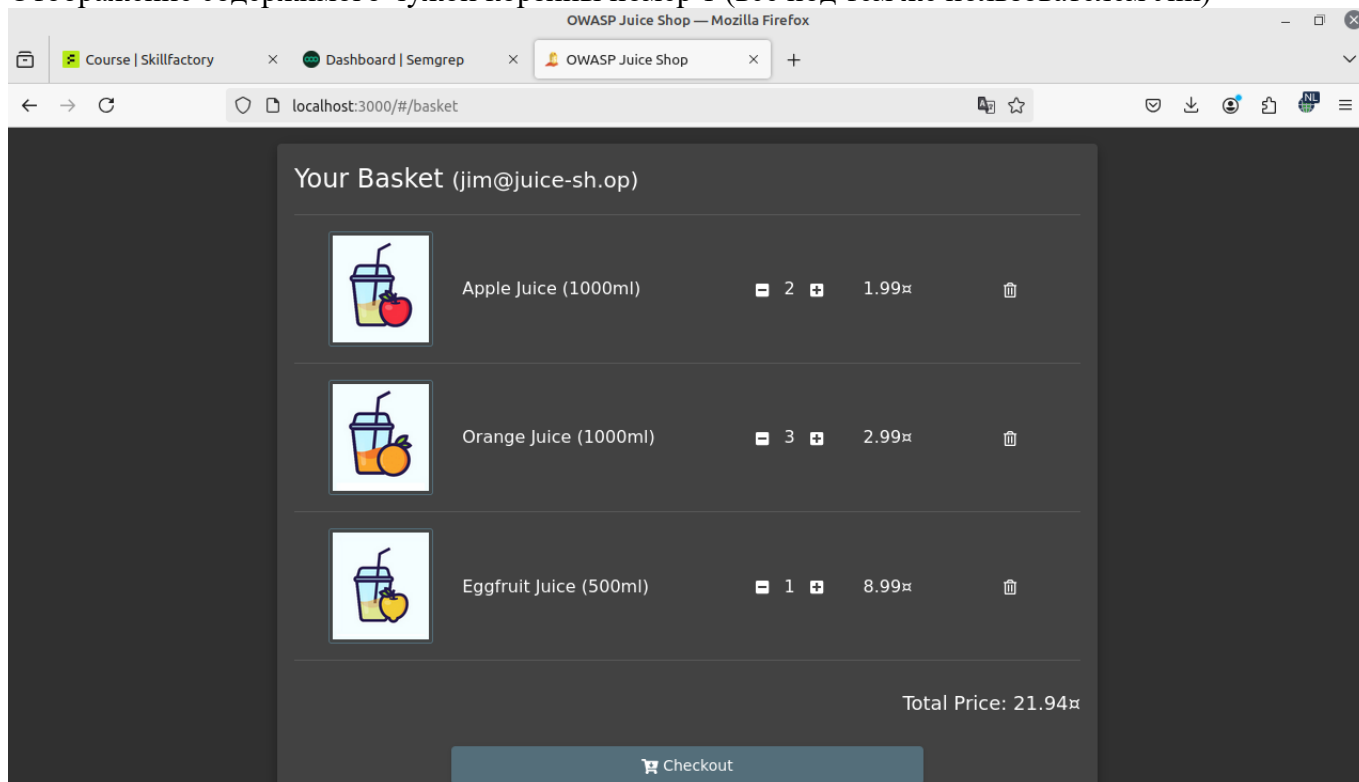
[illegible]



Моя корзина номер 2 (под пользователем Jim)



Отображение содержимого чужой корзины номер 1 (всё под тем же пользователем Jim)



5. Рекомендации по устранению к продемонстрированным уязвимостям.

Database Schema (A03:2021-Injection) - относится к уязвимостям внедрения SQL-кода. Уязвимости внедрения (injection) возникают, когда данные, предоставленные пользователем, обрабатываются небезопасным способом и могут привести к выполнению произвольного кода на сервере. Эта уязвимость возникает, когда приложение не проверяет входные данные перед их использованием в SQL-запросах. Это позволяет злоумышленнику внедрить вредоносный SQL-код, который может быть выполнен сервером базы данных. В результате злоумышленник может получить доступ к конфиденциальной информации или даже выполнить несанкционированные действия, такие как удаление данных или изменение структуры базы данных. Чтобы минимизировать вероятность таких атак, рекомендуется:

- использовать проверку входных данных, когда перед использованием входных данных в SQL-запросе необходимо проверить их на наличие потенциально опасных символов, таких как одинарные кавычки ('), двойные кавычки ("), символы комментария (--) и другие. Если эти символы обнаружены, то они должны быть экранированы или удалены.
- использовать параметризованные запросы, когда вместо того чтобы объединять входные данные с SQL-запросом, следует использовать параметризованные запросы. Параметризованные запросы позволяют отделить данные от запроса, что делает его более безопасным.

View Basket (A01_2021-Broken_Access_Control) – относится к уязвимости контроля доступа, связанной с просмотром корзины покупок, пользователем, не имеющим соответствующих прав доступа. Возникает, когда система не обеспечивает надлежащую защиту от несанкционированного доступа к данным или функциям. Это позволяет злоумышленнику получить доступ к конфиденциальной информации, такой как товары в корзине, количество товаров и общая стоимость покупки. В результате злоумышленник может использовать эту информацию для мошенничества, кражи личных данных или других незаконных действий.

Рекомендации по устранению:

- Ограничить доступ к корзине покупок только для авторизованных пользователей. Это можно сделать, используя механизмы аутентификации и авторизации, такие как логины и пароли, двухфакторная аутентификация или другие методы проверки подлинности.
- Разделить роли пользователей на основе их прав доступа. Например, администраторы могут иметь полный доступ ко всем функциям системы, включая просмотр корзин других пользователей, а обычные пользователи могут иметь ограниченный доступ только к своей корзине.
- В настройках контроля доступа установить разрешения на уровне объектов, не давая другому пользователю создавать, читать, удалять записи.
- Шифрование данных о корзине покупок, чтобы предотвратить их несанкционированный доступ и использование.
- Контроль доступа на уровне API: Если система использует API для взаимодействия с корзиной покупок, убедиться, что запросы на просмотр корзины содержат соответствующие разрешения.

Login Admin (A03:2021-Injection) - Эта уязвимость возникает, когда приложение не проверяет входные данные перед их использованием в SQL-запросах. Это позволяет злоумышленнику внедрить вредоносный SQL-код, который может быть выполнен сервером базы данных. В результате злоумышленник может получить доступ к конфиденциальной информации, получить права администратора, выполнить несанкционированные действия. Основные рекомендации – проверка входных данных, использование параметризованных запросов (более подробно описано по первой уязвимости выше).