Practical Assignment #3

2023/2024

Master in Informatics Engineering
Information Technology Security

Bruno Sequeira 2020235721, brunosequeira@student.dei.uc.pt
Rui Santos 2020225542, rpsantos@student.dei.uc.pt

June 1, 2024

Contents

List of Figures

1 Introduction

In the evolving landscape of web security, the need for robust defensive mechanisms is paramount. This report presents a comprehensive analysis of web application security through the lens of the OWASP Juice Shop, an intentionally vulnerable web application. Our exploration is guided by the Web Security Testing Guide (WSTG) and is executed in two distinct phases, Web application security testing and Web application firewall.

2 Architecture

In the first scenario, we are running kali linux as an attack machine and another one for the webserver. For the second phase we will implement ModSecurity and Apache Reverse Proxy in the webserver machine. Both the attack and webserver machine are Virtual Machines running on Vmware.

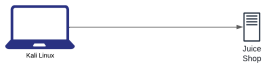


Figure 1: Architecture for phase1

Figure 2: Architecture for phase2

3 Configurations

In order to do this assignment we need to install and configure some software.

3.1 OWASP ZAP

We install ZAP by following the default installation of the software, once it was installed we need to install some addons (those were only installed for the sections about scans with addons). Those addons are: `Active Scanner Rules, Advanced SQLInjection Scanner, Collection, Database, DOM SXX Active Scanner rule` and `Passive Scanner Rules`. We also added some addons to help us doing Fuzz Scans like `Fuzz, FuzzDB Files` and **FuzzDB Offensive**.

ZAP can be run using 4 modes, *Safe Mode, Protected Mode, Standart Mode and Attack Mode*. In Safe Mode, no potentially dangerous operations are permitted, in Protected Mode, we can only perform (potentially) dangerous actions on URLs in the scope, in Standart Mode, does not restrict anything and in Attack Mode, new nodes that are in scope are actively scanned as soon as they are discovered and this can have a destructive effect in the application. In our tests we run ZAP in Standart mode and only changed the policies as we will see later.

3.2 OWASP Juice Shop

For the Juice Shop Web server we used an installation using docker using these commands:

```
sudo apt install docker docker.io
docker pull bkimminich/juice-shop
docker run --rm -p 127.0.0.1:3000:3000 bkimminich/juice-shop
```

3.3 Apache Web Server and WAF

We also need to configure apache to implement WAF and the proxy. First we install WAF and enable some modules that we needed:

```
# WAF installation
sudo su
apt install apache2 libapache2-mod-security2

# Enable some modules that are needed
a2enmod security2 proxy proxy_http proxy_html
```

After this we cloned a repository with many rules to use:

```
git clone https://github.com/coreruleset/coreruleset.git
```

Now we change some file names so the system can recognise them and use them.

```
cd /etc/modsecurity
cp modsecurity.conf-recommended modsecurity.conf
cd coreruleset
mv crs-setup.conf.example crs-setup.conf
```

And after this we can start changing some configurations:

```
# Activate Engine
cd /etc/modsecurity
nano modsecurity.conf

  SecRuleEngine On

# Enable this section and changing paranoia level to 3 in order to make the model more
sensitive
cd /etc/modsecurity/coreruleset
nano crs-setup.conf

  SecAction \
  "id:900000,\
  phase:1,\
  nolog,\
  pass,\
  t:none,\
  setvar:tx.paranoia_level=3"
```

In order to make WAF use the rule set we cloned from github we change this file:

```
nano /etc/apache/mods-available/security2.conf

<IfModule security2_module>
  SecDataDir /var/cache/modsecurity
  IncludeOptional /etc/modsecurity/modsecurity.conf
  IncludeOptional /etc/modsecurity/coreruleset/crs-setup.conf
  IncludeOptional /etc/modsecurity/coreruleset/*.conf
  IncludeOptional /etc/modsecurity/coreruleset/*.load
  SecRuleEngine On
</IfModule>
```

And finally we need do change the apache settings in order to allow it to work as a proxy.

```
nano /etc/apache2/sites-available/000-default.conf

<VirtualHost *:80>
  DocumentRoot /var/www/html
  ErrorLog $APACHE_LOG_DIR/error.log
  CustomLog $APACHE_LOG_DIR/access.log combined
  ProxyPreserveHost On
  ProxyPass / http://localhost:3000/
  ProxyPassReverse / http://localhost:3000/
  <Proxy http://localhost:3000/>
    Require all granted
  </Proxy>
  SecRuleEngine On
</VirtualHost>
```

And finally restart apache2 service to use the new configurations:

```
systemctl restart apache2
```

4 Web application security testing

In this section, we delve into the security posture of the Juice Shop, employing tools like Kali Linux and OWASP ZAP to conduct a series of tests. These tests range from automated and active scans to fuzz attacks and manual penetration testing, all aimed at uncovering potential vulnerabilities.

4.1 Automated Scan

The first test we did to Juice Shop was an automated test. This test is a first step to find vulnerabilities in the different pages of Juice Shop using the tool *Spider*. When we do these tests using OWASP ZAP, it generates a report as shown bellow.

Name	Risk Level	Number of Instances
SQL Injection - SQLite	High	1
Content Security Policy (CSP) Header Not Set	Medium	90
Cross-Domain Misconfiguration	Medium	99
Missing Anti-clickjacking Header	Medium	33
Session ID in URL Rewrite	Medium	131
Cross-Domain JavaScript Source File Inclusion	Low	98
Private IP Disclosure	Low	1
Timestamp Disclosure - Unix	Low	5
X-Content-Type-Options Header Missing	Low	131
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	50
Session Management Response Identified	Informational	27

Figure 3: Automated Scan

As we can see we already detected one High Risk vulnerability, **SQL Injection - SQLite**. This risk is about queries that are not protected and because of that can be easily manipulated.

4.2 Active Scan

Next we did an active scan that is a more aggressive scan do the site. To perform this scan we did some changes in the policies, we set the strength to Insane and the Threshold to low. This allows the attack to be more aggressive.

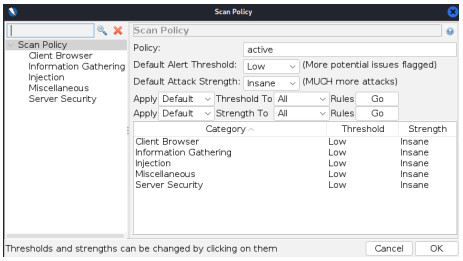


Figure 4: active scan policies

With this we detected a few more vulnerabilities on each risk level. In the High Risk level we detected one more that was **Clout Metadata Potentially Exposed** that is a vulnerability that allows access to cloud metadata because of bad configuration of the server. However, since there is no evidence of Juice Shop using cloud services we don't know if this is a real vulnerability or just a false positive.

Name	Risk Level	Number of Instances
Cloud Metadata Potentially Exposed	High	1
SQL Injection - SQLite	High	8
Content Security Policy (CSP) Header Not Set	Medium	90
Cross-Domain Misconfiguration	Medium	99
ELMAH Information Leak	Medium	1
Missing Anti-clickjacking Header	Medium	33
Session ID in URL Rewrite	Medium	131
Cross-Domain JavaScript Source File Inclusion	Low	98
Private IP Disclosure	Low	1
Timestamp Disclosure - Unix	Low	5
X-Content-Type-Options Header Missing	Low	131
.env Information Leak	Informational	3
.htaccess Information Leak	Informational	3
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	50
Session Management Response Identified	Informational	27
Trace.axd Information Leak	Informational	3
User Agent Fuzzer	Informational	150

Figure 5: active scan policies

4.3 Using Addons in the scans

The use of addons in the active scan increase considerably the number of detected vulnerabilities in all the risk levels.

Name	Risk Level	Number of Instances
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause	Alto	1
Cloud Metadata Potentially Exposed	Alto	1
Open Redirect	Alto	1
SQL Injection - SQLite	Alto	1
Backup File Disclosure	Médio	31
Bypassing 403	Médio	5
CORS Misconfiguration	Médio	129
Content Security Policy (CSP) Header Not Set	Médio	117
Cross-Domain Misconfiguration	Médio	96
Missing Anti-clickjacking Header	Médio	60
Session ID in URL Rewrite	Médio	202
Web Cache Deception	Médio	5
Cross-Domain JavaScript Source File Inclusion	Baixo	98
Dangerous JS Functions	Baixo	2
Deprecated Feature Policy Header Set	Baixo	61
Full Path Disclosure	Baixo	5
Permissions Policy Header Not Set	Baixo	60
Private IP Disclosure	Baixo	1
Timestamp Disclosure - Unix	Baixo	5
X-Content-Type-Options Header Missing	Baixo	202
Base64 Disclosure	Informational	5
CORS Header	Informational	4
Cookie Slack Detector	Informational	16
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	50
Non-Storable Content	Informational	62
Sec-Fetch-Dest Header is Missing	Informational	123
Sec-Fetch-Mode Header is Missing	Informational	123
Sec-Fetch-Site Header is Missing	Informational	123
Sec-Fetch-User Header is Missing	Informational	348
Storable and Cacheable Content	Informational	212
Storable but Non-Cacheable Content	Informational	80
User Agent Fuzzer	Informational	136

Figure 6: active scan with addons

4.4 Scan authenticated

In order to test some areas that might not be tested before, we configured ZAP so it can use credentials to authenticate to Juice Shop as an admin. After configuring this, we did the active scan

again using those credentials and ZAP detected new vulnerabilities, specialty some new types of sql injection and a Source Code Disclosure vulnerability.

Name	Risk Level	Number of Instances
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause	Alto	2
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause (MySQL comment)	Alto	1
Advanced SQL Injection - PostgreSQL boolean-based blind - Stacked queries	Alto	1
Cloud Metadata Potentially Exposed	Alto	1
SQL Injection - SQLite	Alto	5
Source Code Disclosure - File Inclusion	Alto	1
CSP: Wildcard Directive	Médio	5
Content Security Policy (CSP) Header Not Set	Médio	914
Cross-Domain Misconfiguration	Médio	979
ELMAH Information Leak	Médio	1
Hidden File Found	Médio	46
Missing Anti-clickjacking Header	Médio	29
Session ID in URL Rewrite	Médio	79
Application Error Disclosure	Baixo	6
Cross-Domain JavaScript Source File Inclusion	Baixo	1730
Dangerous JS Functions	Baixo	2
Deprecated Feature Policy Header Set	Baixo	890
Full Path Disclosure	Baixo	11
Permissions Policy Header Not Set	Baixo	28
Private IP Disclosure	Baixo	1
Timestamp Disclosure - Unix	Baixo	5
X-Content-Type-Options Header Missing	Baixo	77
.env Information Leak	Informational	2
.htaccess Information Leak	Informational	2
Authentication Request Identified	Informational	2
Base64 Disclosure	Informational	10
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	866
Non-Storable Content	Informational	49
Sec-Fetch-Dest Header is Missing	Informational	940
Sec-Fetch-Mode Header is Missing	Informational	940
Sec-Fetch-Site Header is Missing	Informational	940
Sec-Fetch-User Header is Missing	Informational	1071
Session Management Response Identified	Informational	12
Storable and Cacheable Content	Informational	95
Storable but Non-Cacheable Content	Informational	940
Trace.axd Information Leak	Informational	2
User Agent Fuzzer	Informational	157

Figure 7: active scan authenticated

4.5 Fuzz attacks

In order to detect more vulnerabilities we used ZAP to perform attacks in the login form. To help us with this we installed more addons: `Fuzz`, `FuzzDB Files` and `FuzzDB Offensive`

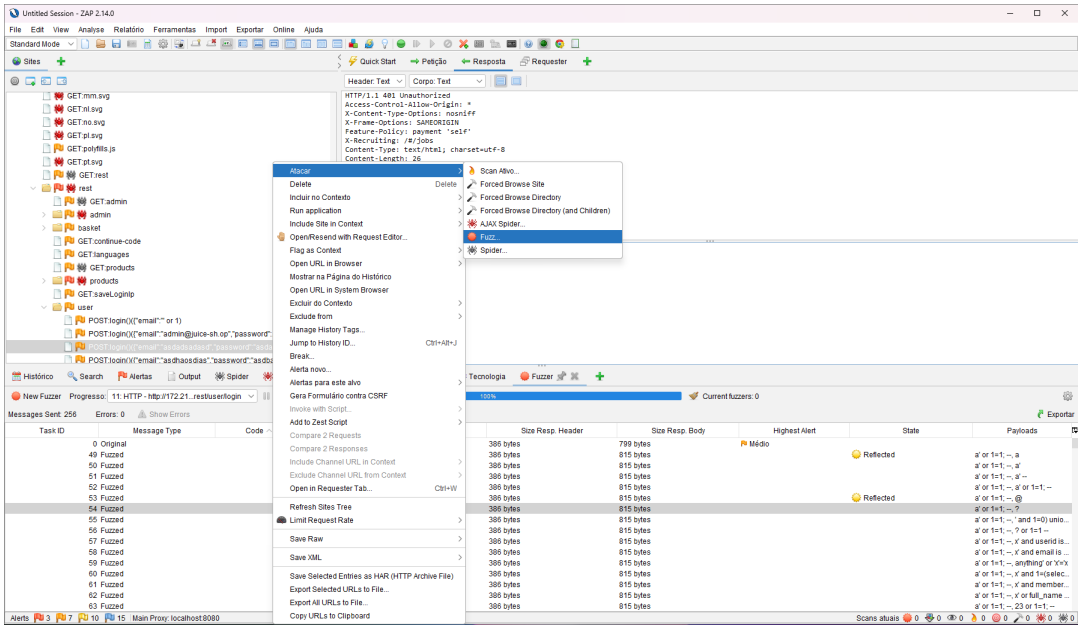


Figure 8: fuzz attack 1

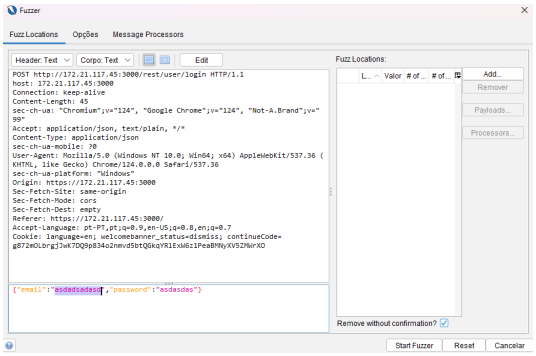


Figure 9: fuzz attack 2

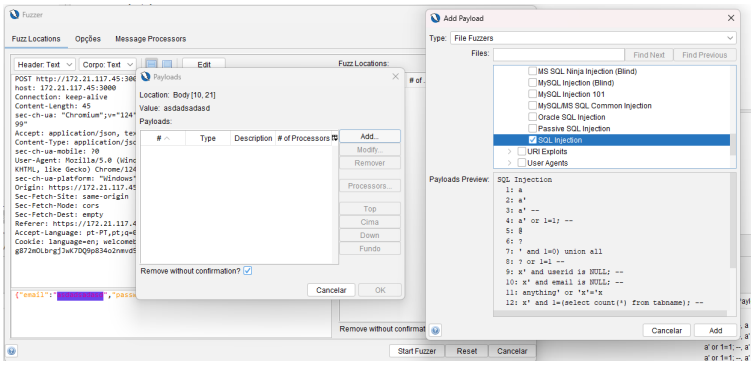
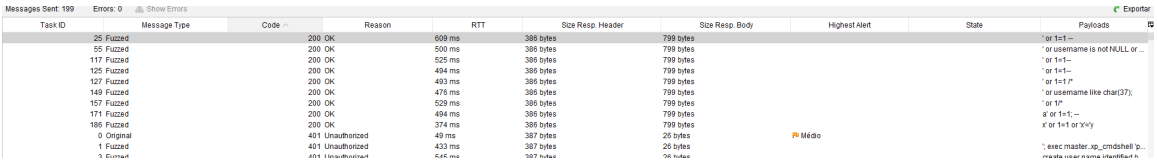


Figure 10: fuzz attack 3

And as a result of the attack we got multiple successful attacks:



Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
25	Fuzzed	200	OK	609 ms	386 bytes	799 bytes			' or 1=1 --
55	Fuzzed	200	OK	500 ms	386 bytes	799 bytes			' or username is not NULL or ...
117	Fuzzed	200	OK	525 ms	386 bytes	799 bytes			' or 1=1--
125	Fuzzed	200	OK	494 ms	386 bytes	799 bytes			' or 1=1--
127	Fuzzed	200	OK	493 ms	386 bytes	799 bytes			' or 1=1 /*
149	Fuzzed	200	OK	476 ms	386 bytes	799 bytes			' or username like char(37);
157	Fuzzed	200	OK	529 ms	386 bytes	799 bytes			' or 1/*
171	Fuzzed	200	OK	494 ms	386 bytes	799 bytes			a' or 1=1; --
186	Fuzzed	200	OK	374 ms	386 bytes	799 bytes			x' or 1=1 or 'x'='y
0	Original	401	Unauthorized	49 ms	387 bytes	26 bytes	Médio		
1	Fuzzed	401	Unauthorized	433 ms	387 bytes	26 bytes			'; exec master..xp_cmdshell 'p...

Figure 11: successful attack

And looking at the responses of those attacks, we now know the email of the admin.


```
{
  "authentication" : {
    "token" :
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiI9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjk1ZjE4YjUwMCIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZG1pbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMDUtMjUgMjE6NDA6NDcuMjU1ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMDUtMjUgMjE6NDA6NDcuMjU1ICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcxNjY3MzI5OH0.hogmYe3Hcn3dPIMhS2in3msBiXHFa2cfXjB2QmAsLuHA9_eQALiXiUO5adoiWHabBjY6vdAqXhSa72b__CgxkhJ7Sa-UjTD53j6UnGhpyQ2Ux9NXaODSINadYaYla_I8vyxviY8B-1pw3yk4hBDEp1Xdw7H-QTb18F5vFL2Vs-Ig",
    "bid" : 1,
    "umail" : "admin@juice-sh.op"
  }
}
```


Figure 12: successful attack

Knowing this, we tried to get the admin password using again a fuzz attack.

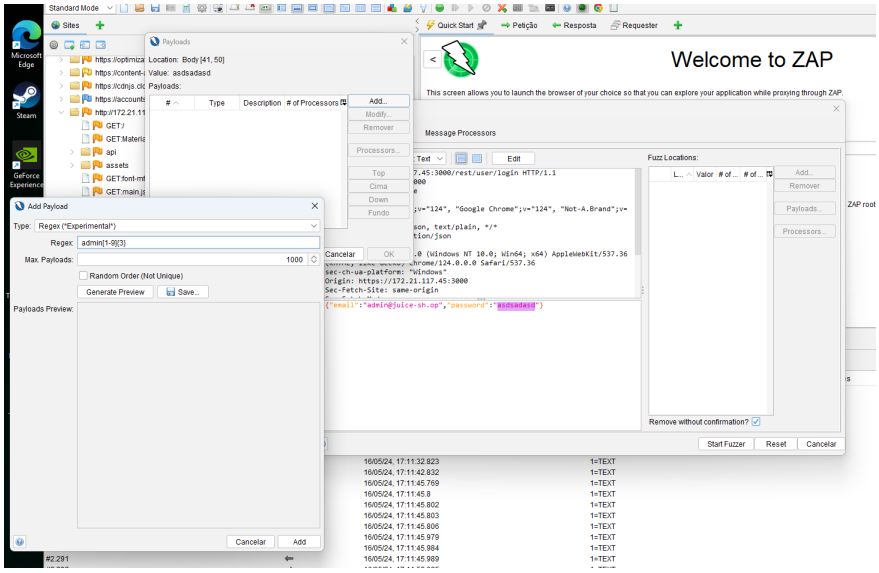


Figure 13: fuzz attack to get password

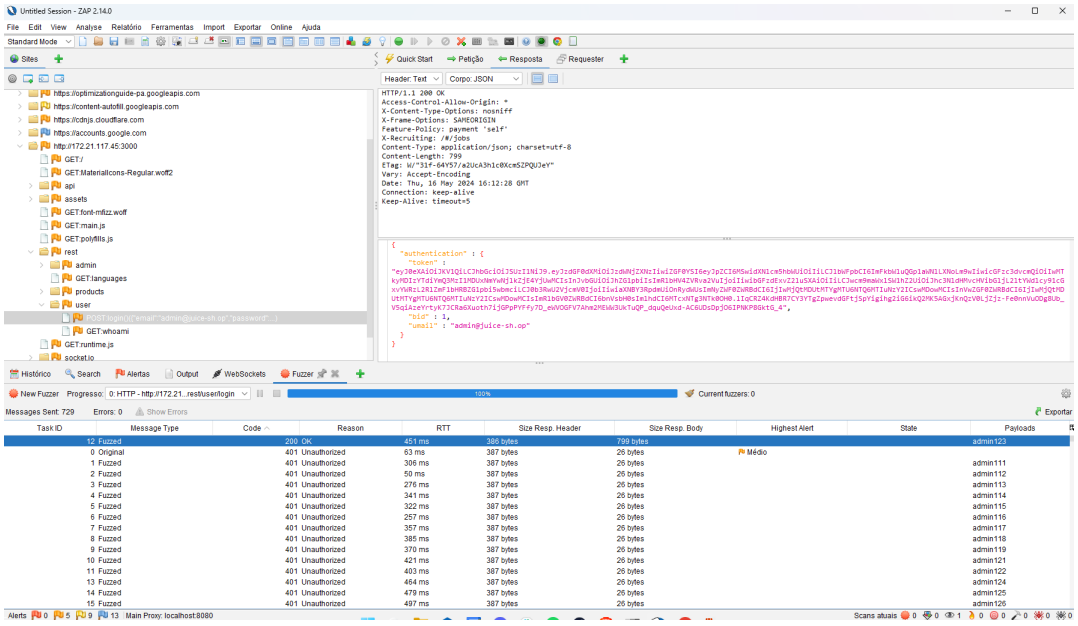


Figure 14: fuzz attack to get password

And we successfully got the admin credentials to Juice Shop that are email: `admin@juice-sh.op` and password: `admin123`.

4.6 Information Gathering

This section focuses on collecting data about the target application and its environment, such as domain names, IP addresses, technologies, versions, and potential vulnerabilities. The goal is to map the attack surface and identify potential entry points for further testing.

4.6.1 WSTG-INFO-01: Conduct Search Engine Discovery Reconnaissance for Information Leakage

This process involves using search engines to search various types of information directly or using

third-party services. Since this web application is not deployed on the Internet and so, it is not possible for the search engines to *crawl* our web app. That said, this subsection is **not applicable**.

4.6.2 WSTG-INFO-02: Fingerprinting Web Server

Web server fingerprinting is the task of identifying the type and version of web server that a target is running on and some relevant information about the web server. For this we use nmap:

```
nmap -A -p 3000 172.21.117.45
```

As we can see, we can get information about the HTTP requests, however we could not identify the web server running the app.

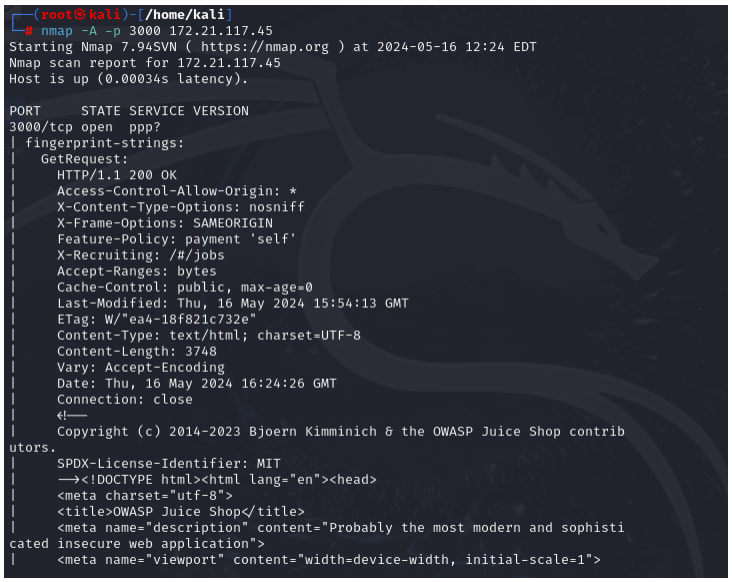


Figure 15: Fingerprint Web Server - nmap command

4.6.3 WSTG-INFO-03: Review Webserver Metafiles for Information Leakage

Here we try to discover and test various metadata files for information leakage, so we try to identify hidden or obfuscated paths and functionality through the analysis of metadata files and try to extract and map other information that could lead to better understanding of the systems at hand. To test this we ran a fuzz attack with some common file names used for metadata.

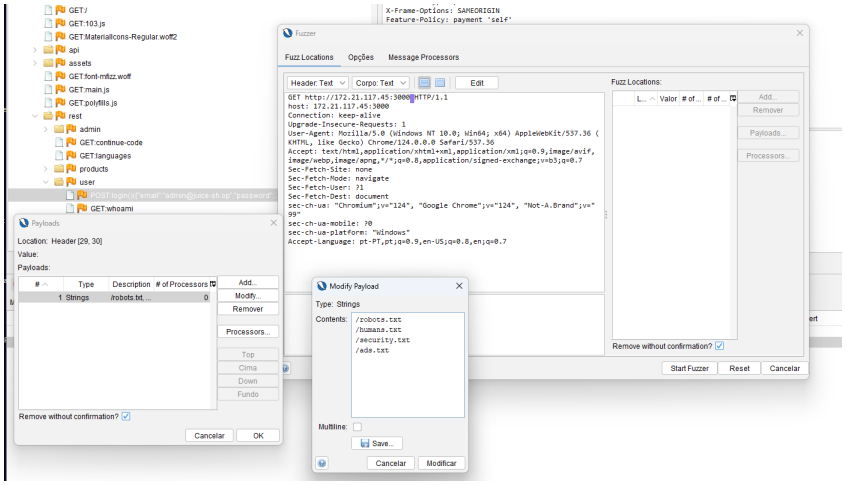


Figure 16: Fuzz attack

Figure 17: robots.txt

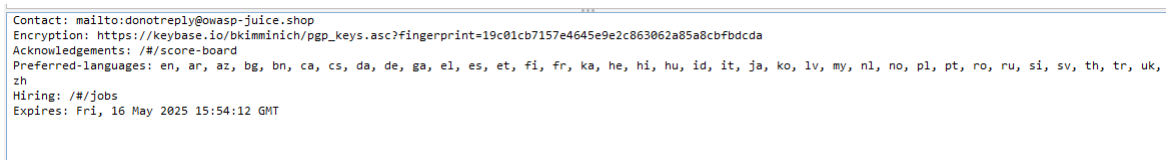


Figure 18: security.txt

Robots.txt and *security.txt* were the files that gave more information. *Robots.txt* contains a list of paths inside the application that we don't want crawlers to access it, in this case we have */ftp*. The *security.txt* is an accepted standard for website security information that allows security researchers to report security vulnerabilities easily.

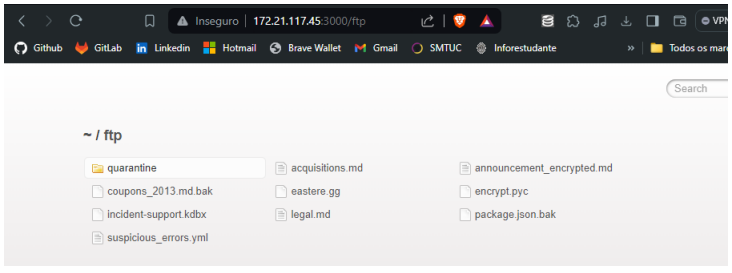


Figure 19: ftp page

4.6.4 WSTG-INFO-04: Enumerate Applications on Webserver

In this topic we want to enumerate the applications within scope that exist on a web server. Those

services can be used to start an attack. In order to find those services we used ZAP.

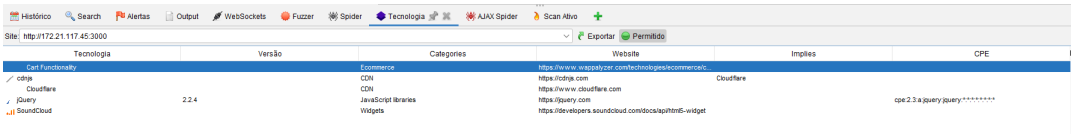


Figure 20: Applications and services associated to Juice Shop

As we can see, we found services *cdnjs, Cart Functionality, Cloudflare, jQuery* and *SoundCloud* that were running in the same machine.

4.6.5 WSTG-INFO-05: Review Webpage Content for Information Leakage

Analyzing the source code of a web application can pose a security risk as it exposes its potential weaknesses to attackers. Therefore, it is crucial to ensure that, at the very least, sensitive information that, if seen by attackers, could guide their attack towards vulnerable areas is not included in the source code. During our analysis of the web app using ZAP, we found that its regular expressions enabled it to detect some instances where there are comments in the source code that are suspiciously likely to contain information that could facilitate an attack. The evidence found by ZAP is presented in the image below.

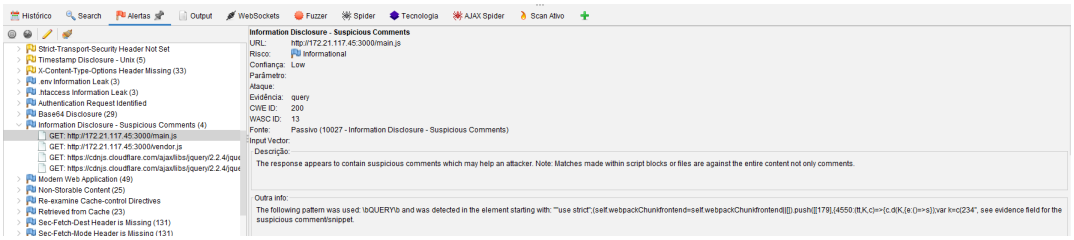


Figure 21: Information Leakage

4.6.6 WSTG-INFO-06: Identify Application Entry Points

This section aims to help identify and map out areas within the application that should be investigated once enumeration and mapping have been completed. Before any testing begins, the tester should always get a good understanding of the application and how the user and browser communicates with it.

To start it might be a good idea to understand what are the HTTP methods used in the web app:

```
nmap -p 3000 --script http-methods 172.21.117.45 -sV
```

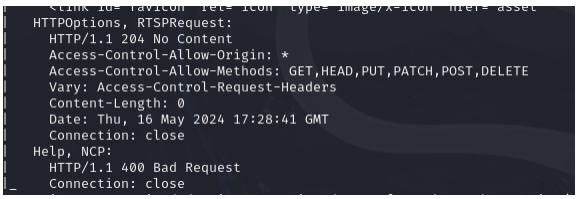


Figure 22: nmap command to see HTTP methods suported

After this we can do some analysis and see the parameters of the requests and the responses. Those can be presented as points of injection since they are easily to manipulate. Consequently, it becomes evident that the Juice Shop has a some entry points, potentially susceptible to vulnerabilities like SQL Injection.

Some examples of endpoints are the login `/rest/user/login` and the product search `/rest/products/search?q=` which have the request parameters q and email and password and the response parameters status and data and token, bid and umail respectively.

Figure 23: login response parameters

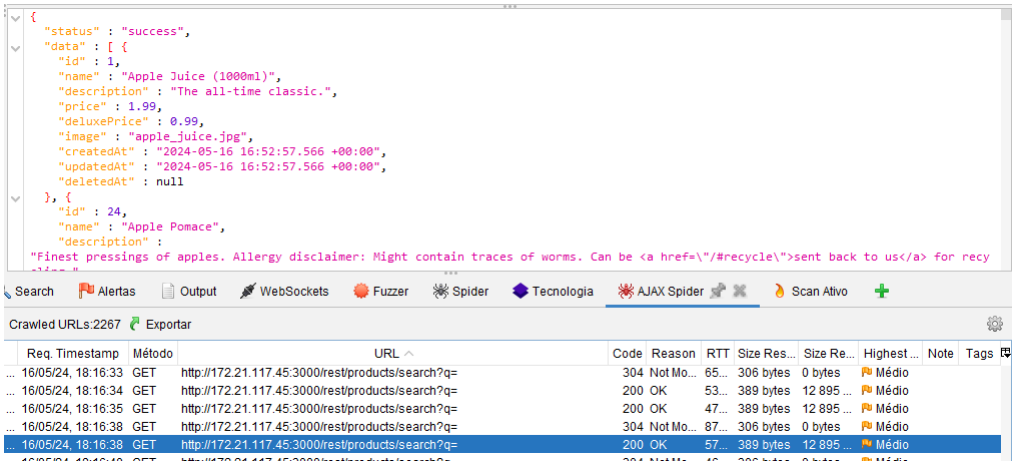


Figure 24: product search response parameters

4.6.7 WSTG-INFO-07: Map Execution Paths Through Application

Without a thorough understanding of the layout of the application, it is unlikely that it will be tested thoroughly. The thorough mapping of various possible pathways within an application is crucial as it allows us to understand the available resources and which of them may be more susceptible or vulnerable to attacks. Therefore, it is imperative to test not only the various possible paths within the application but also the data flows and any potential race conditions that may occur in order to minimize the application's attack risks.

To do this we used ZAP with Spider to crawl through the multiple endpoints. However, we can go deeper by by using ZAP' Force Browsing Site tool with the addon *Directory List*.

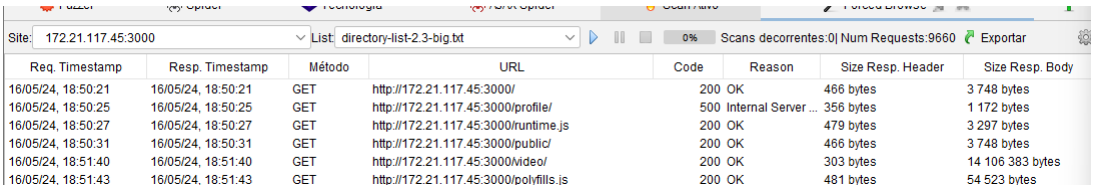


Figure 25: Some results of the Force Browsing Tool

4.6.8 WSTG-INFO-08: Fingerprint Web Application Framework

Nearly every web application that one may think of developing has already been developed. It is very likely that an application security test will face a target that is entirely or partly dependent on

these well known applications or frameworks. These well known web applications have known HTML headers, cookies, and directory structures that can be enumerated to identify the application.

We already talked about part of this topic before but what what we did not talked about yet are cookies.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partitio...	Priority
continueCode	bXYpM3JL9EWR647nV...	172.21.117.45	/	2025-05-16T17:01:31.000Z	72					Medium
cookieconsent_status	dismiss	172.21.117.45	/	2025-05-16T16:56:53.000Z	27					Medium
language	en	172.21.117.45	/	2025-05-16T16:09:41.000Z	10					Medium
token	eyJ0eXAiOiJKV1QiLCJh...	172.21.117.45	/	2024-05-17T01:01:01.000Z	737					Medium
welcomebanner_status	dismiss	172.21.117.45	/	2025-05-16T16:09:57.000Z	27					Medium

Figure 26: Cookies

Ideally, cookies should not have a long expiration date, which is not the case for most of those present in this site as most of them have 1 year expiration date.

4.6.9 WSTG-INFO-09: Fingerprint Web Application

The guidelines say this content has been merged into: Fingerprint Web Application Framework so there is no point of reevaluate.

4.6.10 WSTG-INFO-10: Map Application Architecture

WSTG-INFO-10 focuses on understanding the architecture of web applications and the infrastructure they rely on. Mapping the application architecture is essential to identify potential vulnerabilities that could compromise the entire system. We already covered most of this before so there is no point of reevaluate.

4.7 Configuration and Deployment Management Testing

This category focuses on examining the configuration and deployment settings of the application and its components, including servers, databases, and web services. The objective is to detect any misconfigurations, insecure defaults, or outdated software that could jeopardize the application's security.

4.7.1 WSTG-CONF-01: Test Network Infrastructure Configuration

This test aims to review and validate the configurations of applications across a network, ensuring they are not vulnerable to known flaws. The goal is to ensure that the frameworks and systems used are secure, properly maintained, and do not use default settings or credentials that could be exploited by attackers. In this section we are going to focus on the endpoint `/metrics`. It is possible to have access to the configurations of the web server. For example we can see that the web site is running on nodejs version 21.7.3.

```
# HELP nodejs_heap_space_size_available_bytes Process heap space size available from Node.js in bytes.
# TYPE nodejs_heap_space_size_available_bytes gauge
nodejs_heap_space_size_available_bytes{space="read_only",app="juiceshop"} 0
nodejs_heap_space_size_available_bytes{space="new",app="juiceshop"} 6894712
nodejs_heap_space_size_available_bytes{space="old",app="juiceshop"} 1735592
nodejs_heap_space_size_available_bytes{space="code",app="juiceshop"} 336928
nodejs_heap_space_size_available_bytes{space="shared",app="juiceshop"} 0
nodejs_heap_space_size_available_bytes{space="new_large_object",app="juiceshop"} 8388608
nodejs_heap_space_size_available_bytes{space="large_object",app="juiceshop"} 0
nodejs_heap_space_size_available_bytes{space="code_large_object",app="juiceshop"} 0
nodejs_heap_space_size_available_bytes{space="shared_large_object",app="juiceshop"} 0

# HELP nodejs_version_info Node.js version info.
# TYPE nodejs_version_info gauge
nodejs_version_info{version="v21.7.3",major="21",minor="7",patch="3",app="juiceshop"} 1

# HELP nodejs_gc_duration_seconds Garbage collection duration by kind, one of major, minor, incremental or weakcb.
# TYPE nodejs_gc_duration_seconds histogram
nodejs_gc_duration_seconds_bucket{le="0.001",kind="incremental",app="juiceshop"} 3
nodejs_gc_duration_seconds_bucket{le="0.01",kind="incremental",app="juiceshop"} 3
nodejs_gc_duration_seconds_bucket{le="0.1",kind="incremental",app="juiceshop"} 3
nodejs_gc_duration_seconds_bucket{le="1",kind="incremental",app="juiceshop"} 3
nodejs_gc_duration_seconds_bucket{le="2",kind="incremental",app="juiceshop"} 3
```

Figure 27: /metrics endpoint

4.7.2 WSTG-CONF-02: Test Application Platform Configuration

Here we are aiming to ensure that the configuration of the individual elements making up an application architecture is correct, preventing mistakes that might compromise the security of the entire architecture. Configuration review and testing are essential to avoid maintaining generic configurations that are inadequate after installation. Proper configuration of web and application servers is crucial to prevent unnecessary functionalities, such as sample applications, documentation, and test pages, from being active after deployment, as they can be exploited. Removing default and known files, ensuring no debugging code remains in production environments, and properly setting up logging mechanisms are vital for a secure platform.

For this we will pay attention to the complains page `/#/complain`. Here we have a input area saying that only Zip and pdf files are accepted, however it is possible to upload XML files but after that we get a message saying that feature is deprecated, so we have a deprecated API in the production version of the website.

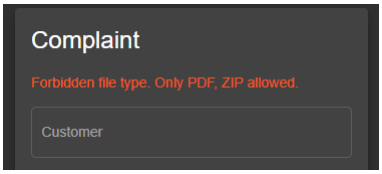


Figure 28: The only types supposed to be accepted

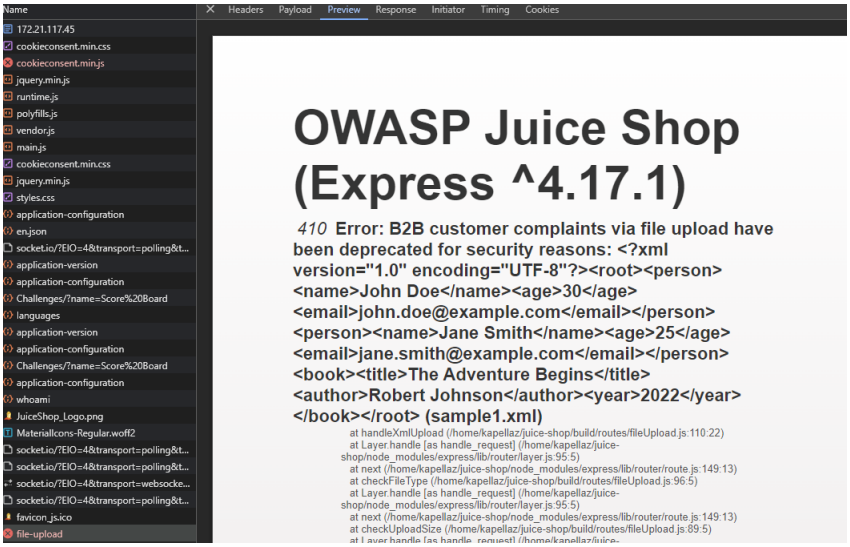


Figure 29: Message after uploading a XML file

4.7.3 WSTG-CONF-03: Test File Extensions Handling for Sensitive Information

No we are trying to ensure that web servers correctly handle file extensions to prevent unintended disclosure of sensitive information or execution of untrusted code. Proper handling of file extensions is crucial as misconfigurations can expose critical details about the server's technologies and access credentials. Here we are going to the complains endpoint and try to upload a file that is not Xml, pdf or zip and see what happends using curl. The endpoint responsible to the file uploads is `/file-upload`.

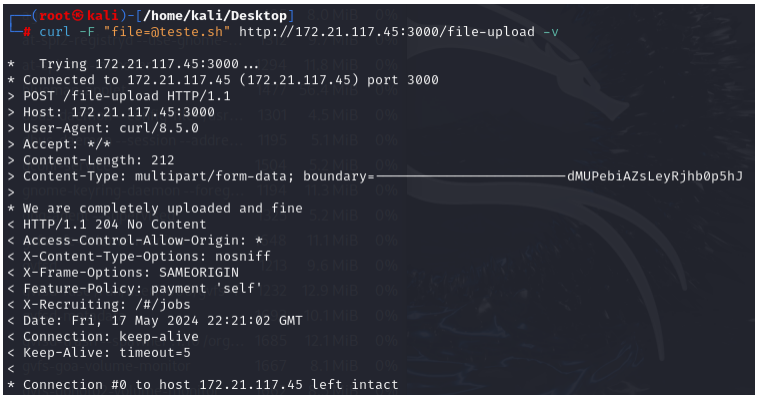


Figure 30: Send a file using curl that don't have the required extension

As we can see, we got the code 204. This code indicates that the request was successful, and the server processed it, but there is no content to send back in the response body. This is something that should not happen.

4.7.4 WSTG-CONF-04: Review Old Backup and Unreferenced Files for Sensitive Information

Now we focus on identifying and analyzing old, backup, and unreferenced files within a web server that may contain sensitive information. These files are often overlooked but can pose significant security risks if accessed by unauthorized users. Going to `/ftp` endpoint we notice we can see .bak files, this file extensions represent backup files and when open they give an error `403 Error: Only .md and .pdf files are allowed!`. With some research we find that there is a method that allow us to see those files even with that error, it is called Poison Null Byte. It allow us to download those files by adding the extension `%2500.md`. Doing this we can actually download the bak files and see their content.

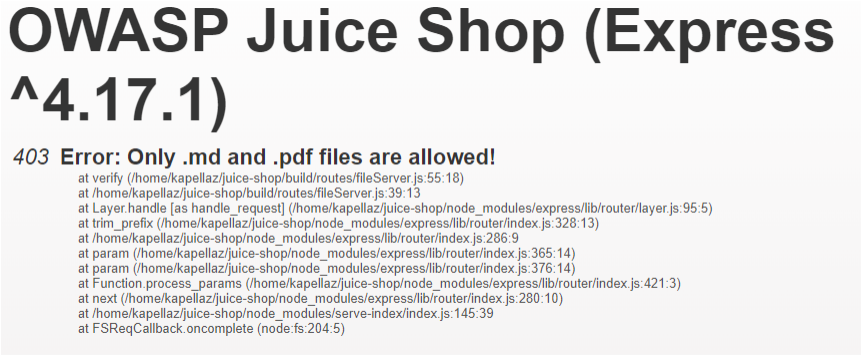


Figure 31: Error accessing the file throw the browser

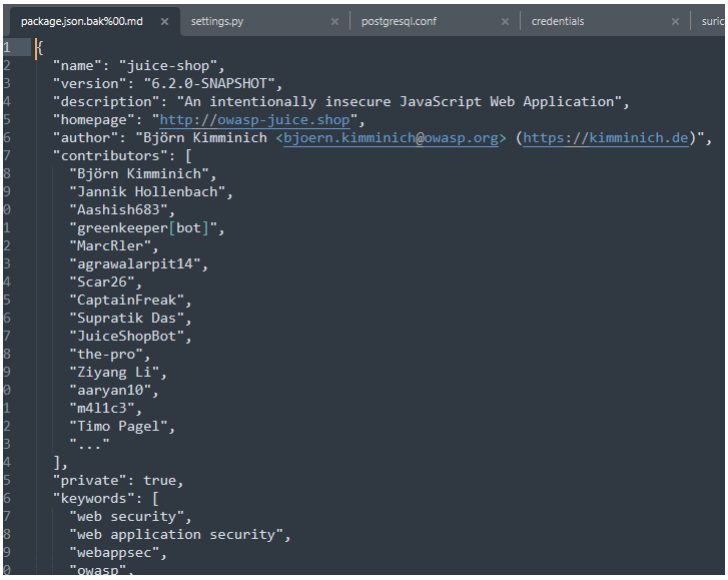


Figure 32: File content after downloading using Poison Null Byte

The file contains some information about a backup configuration file for Juice shop. Despite being forbidden to access throw the website, with a simple research we could find an easy way to bypass it and see the contents in the file.

4.7.5 WSTG-CONF-05: Enumerate Infrastructure and Application Admin Interfaces

Here we try to identify hidden administrator interfaces and functionality within web applications. These interfaces allow privileged users to perform actions such as user account management, site configuration, and data manipulation. Testing focuses on discovering and accessing these interfaces to assess their security controls and prevent unauthorized access. We could identify one endpoint for an admin interface `/#/administration`. Once we discover the admin credentials we can access this page that show us some user information.

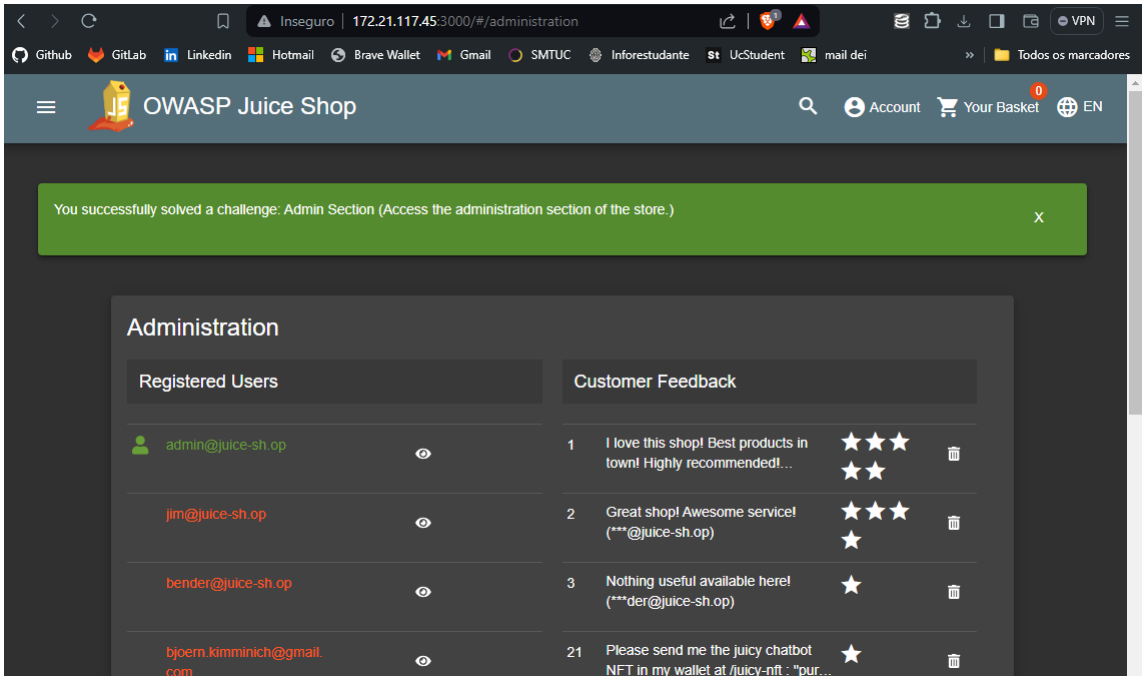


Figure 33: Admin interface

4.7.6 WSTG-CONF-06: Test HTTP Methods

WSTG-CONF-06 focuses on testing various HTTP methods supported by web servers and applications. It aims to identify potential vulnerabilities arising from misconfigurations or improper handling of less common HTTP methods. The objective includes enumerating supported methods, testing for access control bypass, checking for Cross-Site Tracing (XST) vulnerabilities, and evaluating HTTP method overriding techniques. As we saw in WSTG-INFO-06, the site allows the methods GET, HEAD, PUT, PATCH, POST and DELETE.

4.7.7 WSTG-CONF-07: Test HTTP Strict Transport Security

WSTG-CONF-07 focuses on verifying the implementation of HTTP Strict Transport Security (HSTS) on a web application. HSTS is a critical security feature that instructs web browsers to only establish connections with the specified domain via HTTPS, preventing unencrypted HTTP connections. It also mitigates the risk of users overriding certificate errors. Since we are not running Juice Shop using HTTPS this subsection is **not applicable**.

4.7.8 WSTG-CONF-08: Test RIA Cross Domain Policy

Now we try to the configuration of Rich Internet Applications (RIA) cross-domain policy files, such as crossdomain.xml and clientaccesspolicy.xml. These policy files control cross-domain access for technologies like Adobe Flash, Silverlight, and Oracle Java, potentially exposing sensitive data and enabling Cross-site Request Forgery (CSRF) attacks if poorly configured. We don't have this technology in Juice Shop, so this subsection is **not applicable**.

4.7.9 WSTG-CONF-09: Test File Permission

WSTG-CONF-09 aims to evaluate the file permissions set for various resources within a web application's environment. Misconfigured file permissions, granting broader access than necessary, can lead to unauthorized access, data exposure, or manipulation by unintended actors. This test assesses the adequacy of file permissions, especially concerning program configuration, execution files, and sensitive data storage. According to the project statement this subsection is **not applicable**.

4.7.10 WSTG-CONF-10: Test for Subdomain Takeover

This focuses on identifying and mitigating vulnerabilities related to subdomain takeover, where an attacker gains control over a subdomain of a victim's domain. This vulnerability typically arises due to misconfigured DNS settings, such as pointing a subdomain to a non-existing or non-active resource, combined with inadequate subdomain ownership verification by service providers. Subdomain takeover can lead to various malicious activities, including serving malicious content, phishing attacks, and stealing user session cookies or credentials. Since this application is not using DNS and according to the project statement this subsection is **not applicable**.

4.7.11 WSTG-CONF-11: Test Cloud Storage

WSTG-CONF-11 aims to evaluate the security of cloud storage services utilized by web applications and services. Improper access control configurations in cloud storage platforms can lead to various security risks, including unauthorized access, data tampering, and exposure of sensitive information. This test objective primarily focuses on assessing the adequacy of access control configurations to mitigate the risk of unauthorized access and data breaches. Since this application does not use Cloud Storage and according to the project statement this subsection is **not applicable**.

4.8 Identity Management Testing

This category involves testing the application's identity management features, such as user registration, account management, password policies, and profile management. The aim is to identify weaknesses or flaws that could lead to unauthorized access or impersonation.

4.8.1 WSTG-IDNT-01: Test Role Definitions

WSTG-IDNT-01 aims to evaluate the implementation and security of role-based access control (RBAC) within web applications. By defining roles, applications ensure that users can perform tasks relevant to their permissions, thereby maintaining the integrity and security of the application. This objective focuses on identifying and documenting roles, attempting to switch or change roles, and reviewing the granularity and appropriateness of the permissions associated with each role. At first we found only two roles, the costumer and the administrator, however once we login with the admin account, we find that there is a deluxe role as there is a deluxe membership. Also, when looking at the registered accounts in the administration page, we notice two emails, `accountant@juice-sh.op` and `support@juice-sh.op` so we suspected that there might be some accounting and costumer support role in the app and after some research we found `/#/accounting` endpoint but we could not find a costumer support area. Doing some testes we can see that the normal user can't access the admin area or the accounting area, the admin can access his area but not the accounting area. As we did not got a deluxe or accountant credentials yet, we did not test what those roles users can access, but at least for those who we test, everything seems to be as it should.

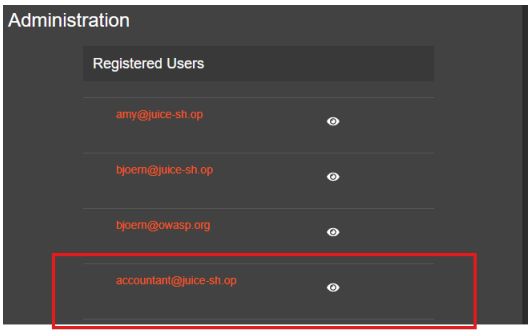


Figure 34: Evidence of an accountant role

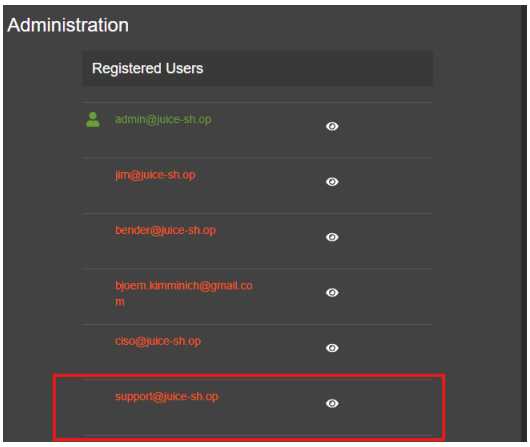


Figure 35: Evidence of a costumer support role

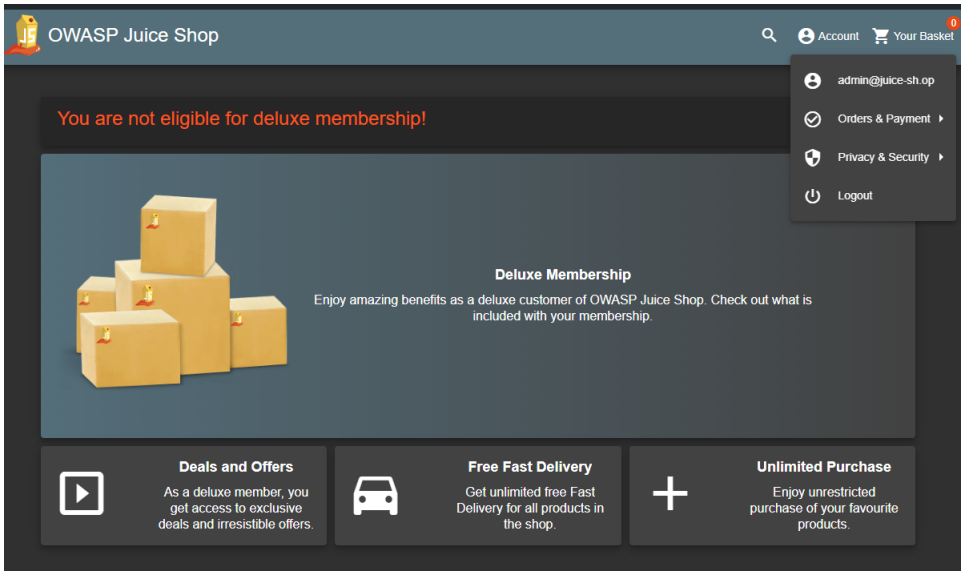


Figure 36: Admin can't access deluxe area

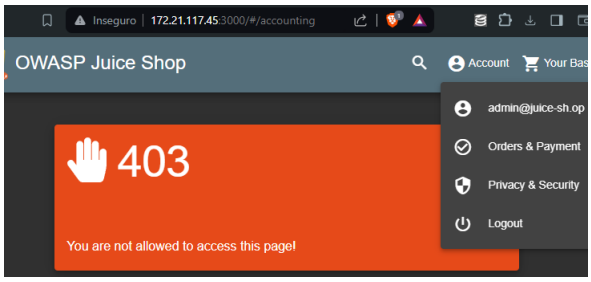


Figure 37: Admin can't access accounting area

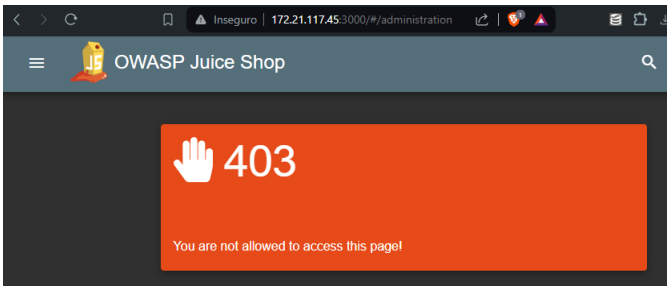


Figure 38: User can't access admin area

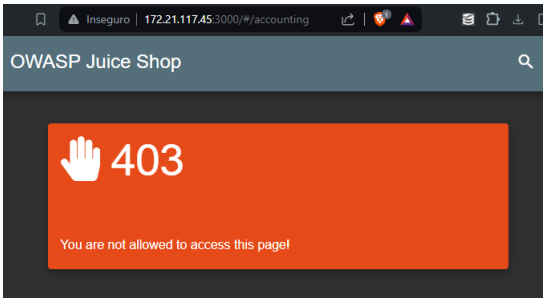


Figure 39: User can't access accounting area

4.8.2 WSTG-IDNT-02: Test User Registration Process

Here we focus on evaluating the user registration process of web applications to ensure that identity requirements are properly aligned with business and security needs. The goal is to verify that the registration process effectively manages and verifies user identities according to the security level of the application. In Juice Shop's registration page the password only requirement is to have at least 5 characters. The minimum of 5 characters is to insecure as most applications require at least 8 characters. To our understanding the password advice section is just something to help to create a stronger password but is not something enforced, as in our test we created an account with the password *teste* and it only follow one advice. After creating that account we tried to login with it and it was successful, so, this application allows simple and not secure passwords.

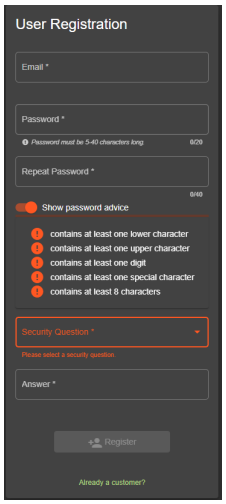


Figure 40: Registration page

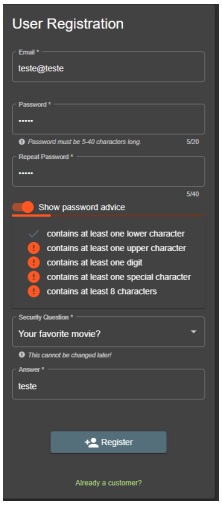


Figure 41: Site allow insecure passwords

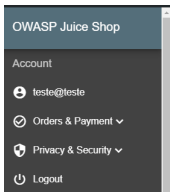


Figure 42: Account with insecure password can login

4.8.3 WSTG-IDNT-03:Test Account Provisioning Process

Here we are aiming to evaluate the account provisioning process to ensure that accounts are created and managed with the proper identification and authorization procedures. This test ensures that unauthorized accounts cannot be created, and that account management follows strict security protocols. As we saw in the previous topic, user registration is not done in the best way. In addiction to the problems discussed before, the application allows an user to use any email, even if it does not exist, like in the example before, in `teste@teste`, the email's domain does not exist.

4.8.4 WSTG-IDNT-04:Testing for Account Enumeration and Guessable User Account

WSTG-IDNT-04 aims to verify if it is possible to collect valid usernames by interacting with the application's authentication mechanism. This information can be used to conduct brute force attacks to guess passwords for these valid usernames. Here we can easily collect usernames since we can see them in the reviews (in this case the email is used, not an username), or, since we already have admin credentials, see them in the administration page. From there we can just try brute forcing and since the site allows 5 character passwords, it is very likely to crack at least one of them or using fuzz attacks like we used to find the admin password.

4.8.5 WSTG-IDNT-05: Testing for Weak or Unenforced Username Policy

This topic aims to determine whether the structure of account names in a web application makes it vulnerable to account enumeration attacks. The goal is to identify if predictable username patterns or error messages can be exploited to discover valid usernames. Since Juice shop uses emails and not usernames this subsection is **not applicable**.

4.9 Authentication Testing

This category examines the authentication mechanisms of the application, including login, logout,

password recovery, and multi-factor authentication. The purpose is to find any weaknesses that could allow bypassing or compromising the authentication process.

4.9.1 WSTG-ATHN-01: Testing for Credentials Transported over an Encrypted Channel

WSTG-ATHN-01 aims to verify that web applications encrypt authentication data in transit. This ensures that credentials and sensitive data are protected from attackers who may intercept network traffic. Proper encryption, typically via HTTPS, is essential for securing data exchanged between clients and servers during login, session management, password resets, and other interactions involving credentials. Since our application is running with no HTTPS support this subsection is **not applicable**.

4.9.2 WSTG-ATHN-02: Testing for Default Credentials

Now we try to identify and verify the presence of default credentials in web applications and their underlying infrastructure. Default credentials, which are often left unchanged after installation, pose significant security risks as they can be easily exploited by attackers to gain unauthorized access. We already test for this before but we only manage to find the admin password using a fuzz attack. However, we can create accounts using default password like we did before for the email `teste@teste` we used the password `teste` which can enable a test for identifiable patterns since the password is the same or close to the username part of the email.

4.9.3 WSTG-ATHN-03: Testing for Weak Lock Out Mechanism

WSTG-ATHN-03 aims to evaluate the effectiveness of account lockout mechanisms in preventing brute force attacks and unauthorized access. Account lockout mechanisms should balance between protecting against unauthorized access and ensuring legitimate users are not denied access. Juice Shop does not present any lock out mechanism and we already saw that when running a fuzz attack to try to get the admin password.

4.9.4 WSTG-ATHN-04: Testing for Bypassing Authentication Schema

Now we focus on evaluating the robustness of authentication mechanisms in web applications. This involves identifying and exploiting weaknesses that allow bypassing authentication controls, thereby gaining unauthorized access to protected resources. We already saw that is possible to bypass the Authentication schema using Sql Injection (`' or 1=1--` and any password is enough to access admin account). However it is also possible to do using the session token. Using ZAP to see the contents of the server requests/responses we can get the token used in plain text as the site is not using HTTPS, with that we used it to send a rating and went to the admin dashboard and verify that it is there and the email used for the rating was the email associated to the token (the email is partialy visible but believe it is `teste@teste`).

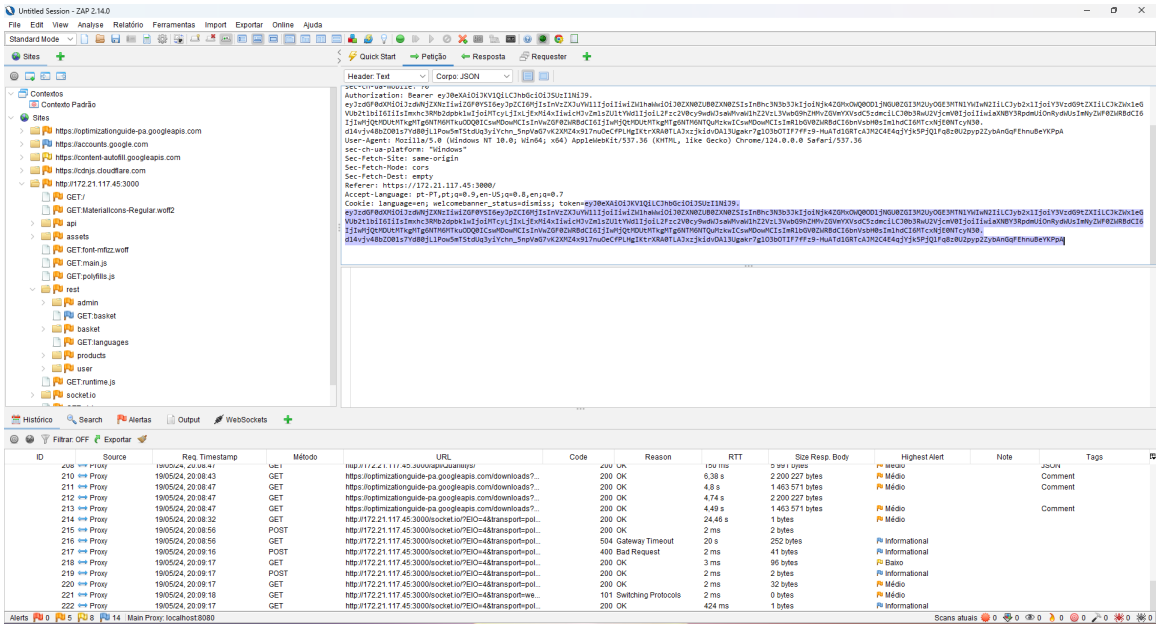


Figure 43: Using ZAP to get the token

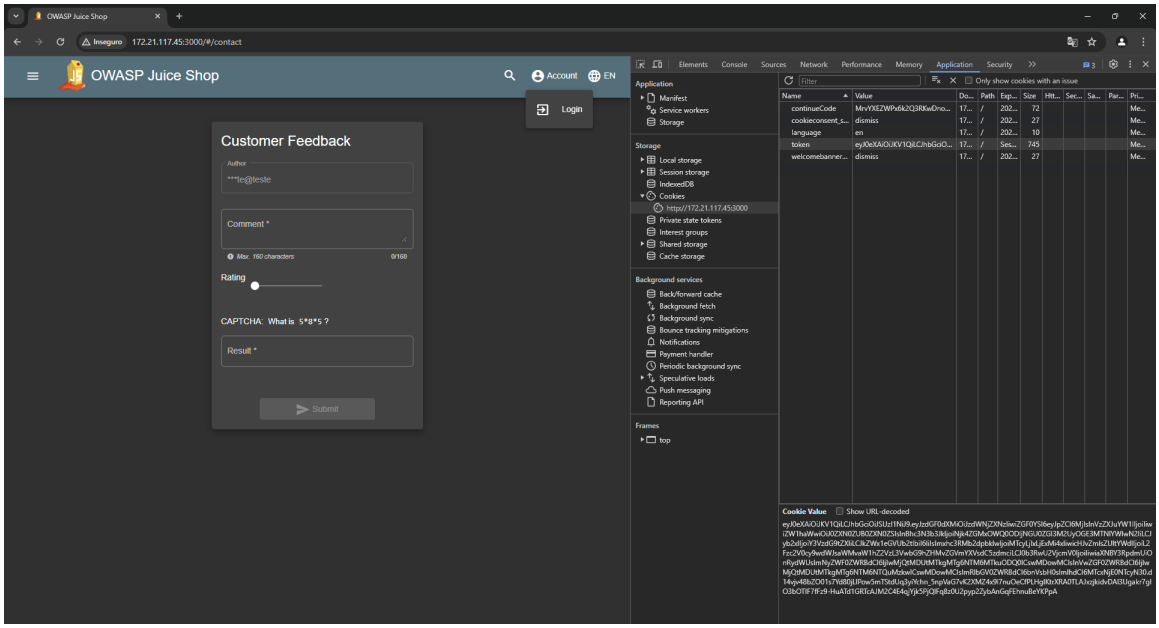


Figure 44: Using the token to send a rating

Figure 45: Rating on the admin page

4.9.5 WSTG-ATHN-05: Testing for Vulnerable Remember Password

WSTG-ATHN-05 aims to validate the security of ”remember me” functionalities and password managers within web applications. These features, designed to enhance user convenience by allowing prolonged authentication sessions, must be assessed for potential security vulnerabilities that could

expose user credentials. For this we try to find what information is stored when the "remember me" option is selected and in the local storage we can see that the email is stored in there, enhancing possible attacks.

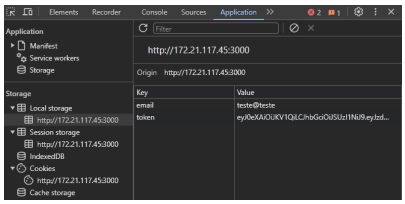


Figure 46: Email address stored in local storage

4.9.6 WSTG-ATHN-06: Testing for Browser Cache Weaknesses

This aims to ensure that web applications do not improperly store sensitive information in the browser's cache or history, which could lead to unauthorized access or data leaks. In the case of Juice Shop, there is little usage of cache and when it is used, it is immediately discarded, guaranteed by the usage of HTTP headers, in this case, Cache-Control that is set to `public, max-age=0`

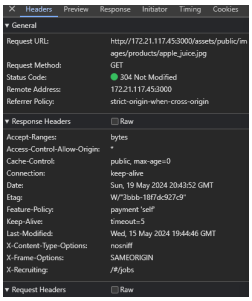


Figure 47: HTTP headers to handle cache

4.9.7 WSTG-ATHN-07: Testing for Weak Password Policy

WSTG-ATHN-07 focuses on evaluating the robustness of an application's password policies to ensure they are resistant to brute force attacks and common password weaknesses. We already talked about part of this topic before in 4.8.2 and 4.8.4 and from that we conclude that this application allows weak passwords. In addiction to that, regarding changing password, there is no limitation when using old passwords.

4.9.8 WSTG-ATHN-08: Testing for Weak Security Question Answer

Here we evaluate the security of the questions and answers used in authentication processes, particularly for password recovery or as an additional layer of security. In this case, it is required to define a security question when setting an account, however there is no possibility of creating our own security questions, and the ones provided are quite known and kind of easy to guess, for example the mother's name question can be easily guessed through social media for example. Also, when trying to guess the security question answer, we test to fail the answer multiple times and did not notice any lock out mechanism, so this can lead to potential attacks.

4.9.9 WSTG-ATHN-09: Testing for Weak Password Change or Reset Functionalities

WSTG-ATHN-09 aims to evaluate the security of an application's password change and reset functionalities, ensuring they resist unauthorized access or manipulation. This topic was already covered before and we concluded that those functionalities have a lot of problems and are not secure.

4.9.10 WSTG-ATHN-10: Testing for Weaker Authentication in Alternative Channel

To conclude the Authentication Testing we are now aiming to identify and assess the security of alternative authentication channels to ensure they do not introduce vulnerabilities that could be exploited to bypass primary authentication mechanisms and since Juice Shop does not have an alternative channel for authentication, this subsection is **not applicable**.

4.10 Authorization Testing

This category focuses on testing the application's authorization functionality, such as role-based access control, privilege escalation, and horizontal and vertical access controls. The goal is to uncover any weaknesses that could permit unauthorized access or manipulation of data and functionality.

4.10.1 WSTG-ATHZ-01: Testing Directory Traversal File Include

WSTG-ATHZ-01 focuses on identifying and mitigating directory traversal and file inclusion vulnerabilities in web applications. These vulnerabilities allow attackers to read or write files outside the intended directory structure, potentially leading to unauthorized access or execution of arbitrary code. We already covered this on 4.7.4 when we use Poison Null Byte to read files that were not intended to.

4.10.2 WSTG-ATHZ-02: Testing for Bypassing Authorization Schema

Now we try to assess the effectiveness of the authorization schema implemented in a web application. It focuses on verifying whether users can bypass access controls to gain unauthorized access to resources or perform actions beyond their assigned privileges. One situation that we found was on the view basket page where we can Bypassing Authorization Schema in order to see other user's baskets. To do this we only need do change the **bid** variable in the browser's session storage

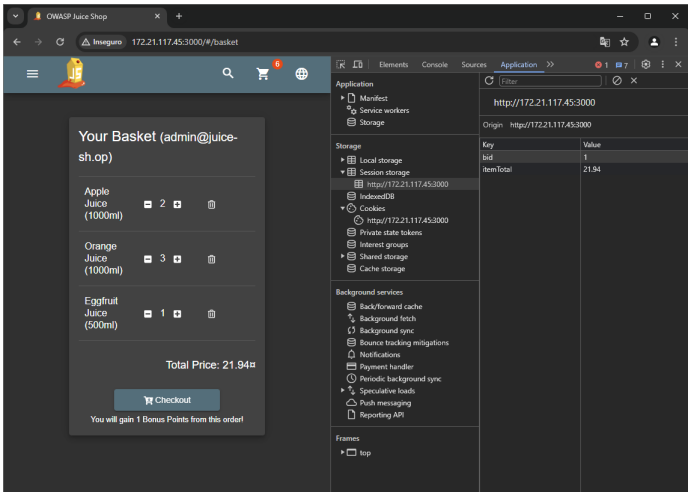


Figure 48: Admin's basket before bid change

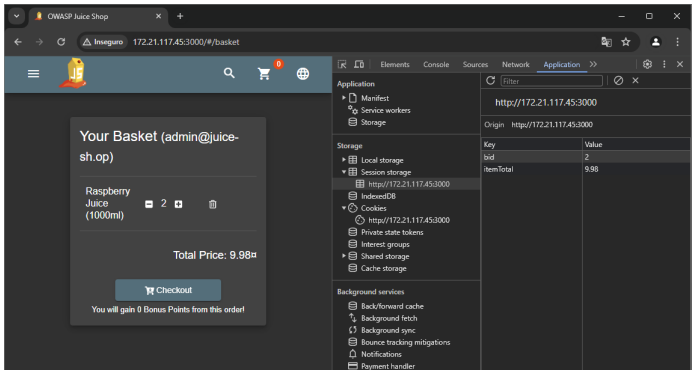


Figure 49: Admin's basket after bid change

4.10.3 WSTG-ATHZ-03: Testing for Privilege Escalation

WSTG-ATHZ-03 aims to assess whether users can manipulate their privileges or roles within the application to escalate their access beyond what is intended. This testing phase focuses on identifying and preventing privilege escalation vulnerabilities, where users gain access to resources or functionality they shouldn't have. This was already verified since we can get admin credentials using Sql Injection, giving admin privileges to an attacker.

4.10.4 WSTG-ATHZ-04: Testing for Insecure Direct Object References

WSTG-ATHZ-04 focuses on identifying and assessing Insecure Direct Object References (IDOR) vulnerabilities within web applications. IDOR vulnerabilities occur when an application allows users to access objects directly based on user-supplied input, bypassing proper authorization checks. Attackers can exploit IDOR vulnerabilities to access unauthorized resources such as database records or files. We already cover this since we prove that with Sql Injection we can manipulate the SqlLite database in order to get access to an admin account.

4.11 Session Management Testing

This category focuses on testing session management features, such as session creation, maintenance, termination, and hijacking. The aim is to identify flaws that could allow attackers to steal or manipulate session tokens or cookies.

4.11.1 WSTG-SESS-01: Testing for Session Management Schema

WSTG-SESS-01 focuses on evaluating the session management mechanisms of web applications to ensure they securely manage and maintain user sessions. The objective is to verify that session tokens (e.g., cookies) are generated securely, are unpredictable, and resist various attacks such as session hijacking and privilege escalation. As we already saw, we can get the session token using ZAP while scanning the requests and responses because the site is not using any encryption, and with that we also saw that we can use to send a rating for example.

4.11.2 WSTG-SESS-02: Testing for Cookies Attributes

Web cookies, often a target for malicious users, are crucial in maintaining the security of web applications. Due to the stateless nature of HTTP, sessions are created and appended to requests, with cookies being the most common session storage mechanism. Cookies can be set via HTTP response headers or JavaScript for various purposes, such as session management, personalization, and tracking.

Securing cookies involves employing various attributes and prefixes to mitigate their vulnerability to attacks. This section outlines the key security attributes and prefixes for cookies and tests to their proper implementation.

The Path that defines the URL path that must exist in the requested resource before sending the cookie restricting the cookie to specific paths, preventing its use across different parts of the application, an in this case is always set to \ and this can cause security vulnerabilities as it is to generic. The Expires Attribute sets the expiration date and time for the cookie determining the cookie's lifespan, differentiating between session and persistent cookies and in the case of Juice Shop some cookies have a long expiration date (1year) which is not recommended. Then we have the HttpOnly attribute that has the purpose of preventing client-side scripts from accessing the cookie. reducing the risk of session leakage and certain types of cross-site scripting (XSS) attacks. We also have Secure and SameSite that are both set to false but should be set to true. SameSite controls whether cookies are sent with cross-site requests and Secure ensures the cookie is sent only over HTTPS.

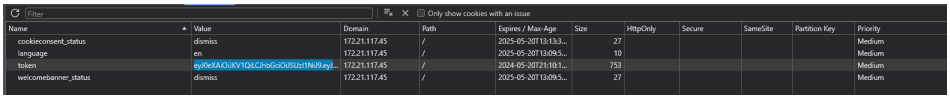


Figure 50: Cookies present in Juice Shop

4.11.3 WSTG-SESS-03: Testing for Session Fixation

WSTG-SESS-03 focuses on identifying and mitigating session fixation vulnerabilities in web applications. Session fixation occurs when a web application preserves the same session identifier before and after user authentication. This can allow an attacker to force a victim to use a known session identifier, which the attacker can then use to hijack the session. In this case we noticed that the session token is clean after the user logs out.

4.11.4 WSTG-SESS-04: Testing for Exposed Session Variables

WSTG-SESS-04 is designed to identify vulnerabilities related to exposed session variables. These vulnerabilities occur when session tokens (cookies, session IDs, hidden fields) are not properly protected during their transmission between the client and the server. Exposed session tokens can enable attackers to impersonate users and gain unauthorized access to web applications. Using ZAP we can see that the scans alerted for Session ID in URL Rewrite, and looking at the packets analyzed we can see the exposed sessionID.

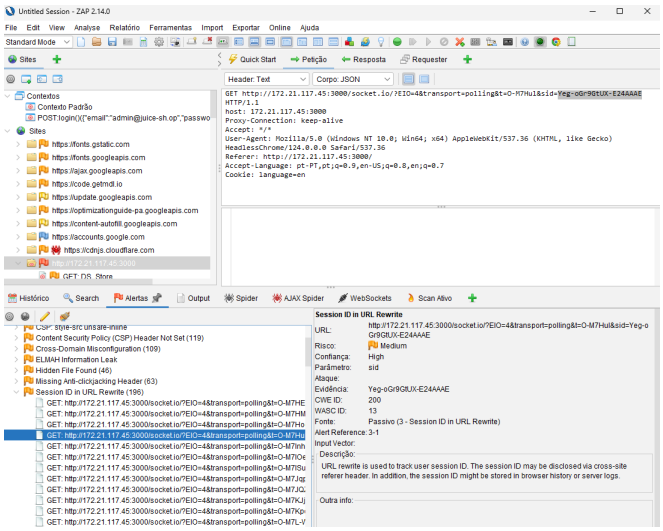


Figure 51: SessionID exposed

4.11.5 WSTG-SESS-05: Testing for Cross Site Request Forgery

Cross-Site Request Forgery (CSRF) is an attack where a user is tricked into executing actions on a web application in which they are authenticated, without their consent. This exploit can compromise

user data and operations. If an administrator account is targeted, the entire application can be compromised. In the case of Juice Shop, we found evidence that no anti CSRF tokens were used, using ZAP.

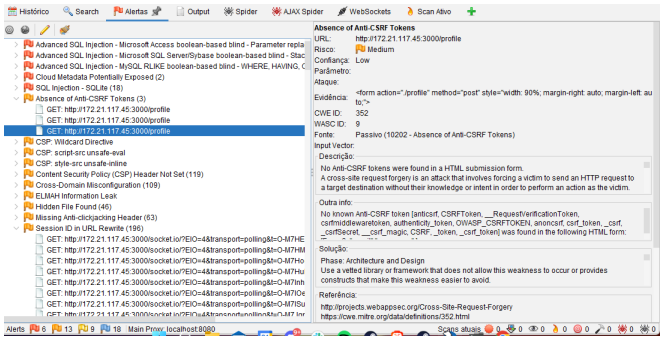


Figure 52: Absence of Anti-CSRF Tokens

4.11.6 WSTG-SESS-06: Testing for Logout Functionality

WSTG-SESS-06 focuses on assessing the logout functionality of a web application. Proper session termination is essential for minimizing the risk of session hijacking and other security threats. The objective is to evaluate the availability and effectiveness of logout controls, session timeout mechanisms, and server-side session termination. We already discuss this before, concluding that the user data is cleaned after logout, except when the remember me button is selected while logging in, in that case, the email is stored in the Local Storage.

4.11.7 WSTG-SESS-07: Testing Session Timeout

WSTG-SESS-07 focuses on verifying that a web application automatically logs out users after a certain period of inactivity, preventing session reuse and eliminating sensitive data stored in the browser cache. This test aims to ensure that session timeout mechanisms are implemented securely, balancing security requirements with usability considerations. In the case of Juice Shop, the only evidence of session timeout is when the token expires, we test this with the EditThisCookie browser extension to set the expiration time to 1 minute and after that all the user data was cleaned.

4.11.8 WSTG-SESS-08: Testing for Session Puzzling

WSTG-SESS-08 aims to detect and mitigate session variable overloading vulnerabilities, also known as Session Puzzling. This vulnerability occurs when an application uses the same session variable for multiple purposes, enabling attackers to manipulate session contexts and potentially bypass authentication mechanisms or elevate privileges. This application only uses the token as session variable and we already prove that an attacker can get the token by analyzing requests and responses to the site, so we believe the site is not safe for session puzzling.

4.11.9 WSTG-SESS-09: Testing for Session Hijacking

WSTG-SESS-09 focuses on identifying and mitigating session hijacking vulnerabilities, which occur when attackers gain access to user session cookies and impersonate them. Session hijacking can lead to unauthorized access to sensitive accounts and data. This test aims to assess the security of session cookies and evaluate the risk level associated with potential hijacking scenarios. We can conclude that the site is not safe against session hijacking since in 4.9.4 we got a token from another user and used it to send a rating.

4.12 Input Validation Testing

This section is about testing the input validation mechanisms of the application, including data sanitization, encoding, and filtering. The objective is to identify vulnerabilities that could allow

the injection or execution of malicious inputs, such as SQL injection, cross-site scripting (XSS), or command injection.

4.12.1 WSTG-INPV-01: Testing for Reflected Cross Site Scripting

WSTG-INPV-01 aims to identify and mitigate Reflected Cross-site Scripting (XSS) vulnerabilities, which occur when attackers inject browser executable code within a single HTTP response. Reflected XSS attacks are non-persistent and rely on tricking users into accessing maliciously crafted links or third-party web pages. The injected attack string is processed improperly by the application and returned to the victim, executing malicious code in their browser. In Juice Shop we identified this vulnerability in the search. By passing the code `<iframe src="javascript:alert(‘teste xss‘)">` and we get the following:

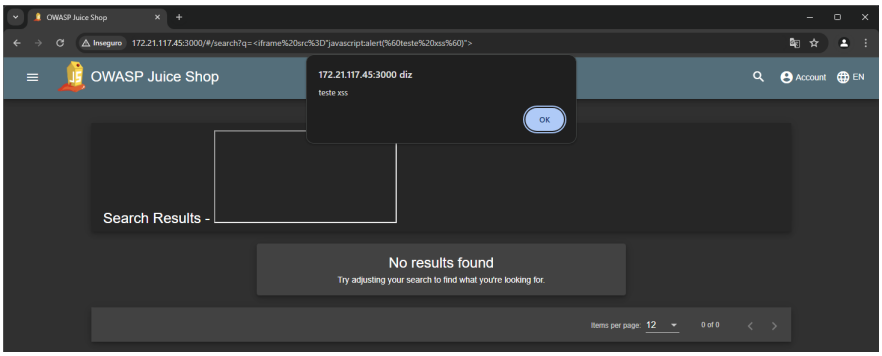


Figure 53: XSS

4.12.2 WSTG-INPV-02: Testing for Stored Cross Site Scripting

WSTG-INPV-02 focuses on detecting and mitigating Stored Cross-site Scripting (XSS) vulnerabilities, which are considered the most dangerous type of XSS. These vulnerabilities arise when web applications allow users to store data that is later displayed to other users without proper filtering. Stored XSS attacks can lead to serious security breaches, enabling attackers to execute malicious code within the context of the application, potentially compromising sensitive information or performing unauthorized actions.

In Juice Shop is possible to store XSS but it needs to be by changing the request using ZAP. In our case we injected XSS using the user registration form. When we sent the information, we find the the request in ZAP and change it to the same code used in the section before and when we access the admin area the code executes.

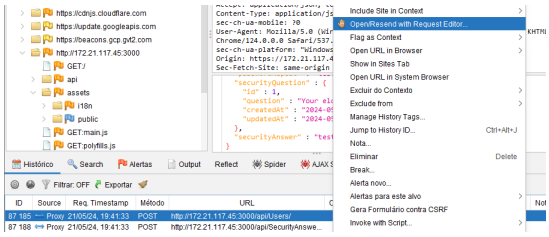


Figure 54: XSS

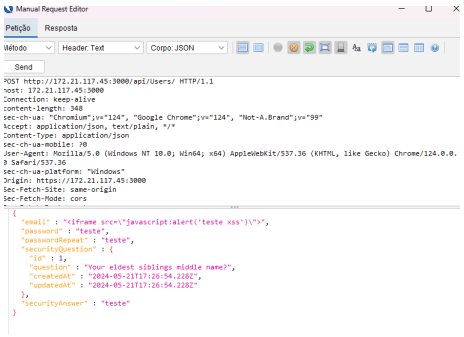


Figure 55: XSS

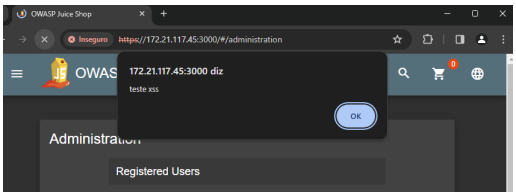


Figure 56: XSS

However, this does not work for every input field. When we did this to create an address, the html code rendered &lt for < and &gt for >, that means that the application might have some protection against stored XSS.

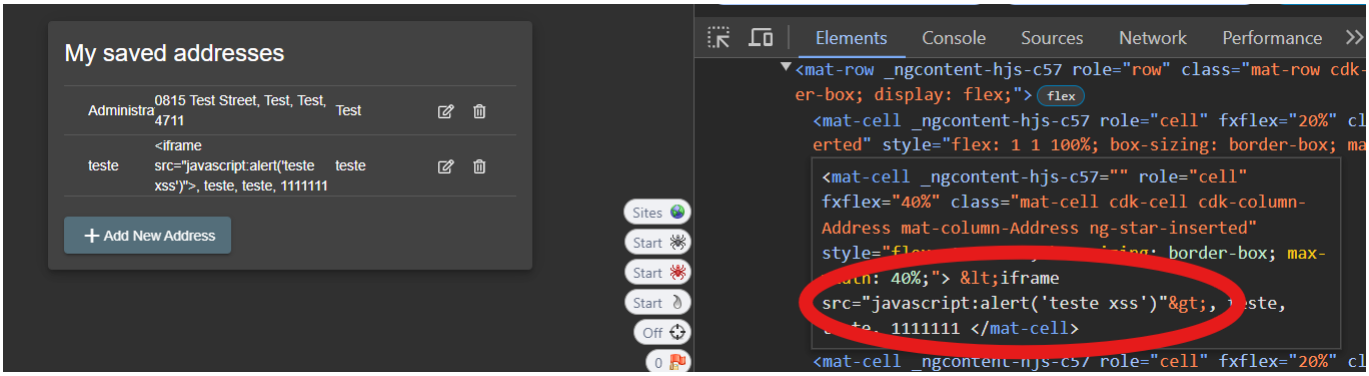


Figure 57: XSS

4.12.3 WSTG-INPV-03: Testing for HTTP Verb Tampering

The guidelines say that this topic was merged with 4.7.6 so we will not discuss this further.

4.12.4 WSTG-INPV-05: Testing for SQL Injection

WSTG-INPV-05 focuses on identifying and mitigating SQL Injection vulnerabilities within web applications. SQL Injection vulnerabilities occur when user inputs are improperly validated and directly incorporated into SQL queries. Exploiting these vulnerabilities allows attackers to execute arbitrary SQL commands, potentially compromising the database and the underlying system. As we already saw, the ZAP scans identified that Juice Shop was vulnerable to sql injection. In adiction to that, we exemplified this vulnerability by using SqlInjection to bypass admin's password.

4.12.5 WSTG-INPV-13: Testing for Format String Injection

WSTG-INPV-13 aims to identify and mitigate Format String Injection vulnerabilities in web applications. These vulnerabilities occur when user input is improperly concatenated with format strings, allowing attackers to manipulate the format specifiers and potentially cause runtime errors, information disclosure, or buffer overflows. We already prove that Juice Shop was vulnerable to this by using poison null byte to get access to forbidden files in 4.7.4.

4.13 Testing for Error Handling

This category examines the application's error handling capabilities, including error messages, logs, and debug information. The goal is to detect weaknesses that could reveal sensitive information to attackers, such as stack traces, database queries, or system details.

4.13.1 WSTG-ERRH-01: Testing for Improper Error Handling

WSTG-ERRH-01 aims to identify and analyze improper error handling in web applications and other systems to prevent information disclosure, system mapping, DoS attacks, and other potential exploits. Proper error handling ensures that applications do not leak sensitive information or expose vulnerabilities through error messages. An example for this is when we tried to login just using " ' " as an email, we got an error that gives information about some of the tables that the Juice Shop database uses. We can also get information from the stack traces that the application generates. We tried this by accessing the page `/ftp/eastere.gg`. These stack traces can be usefull for attackers to understand the structure of the application.

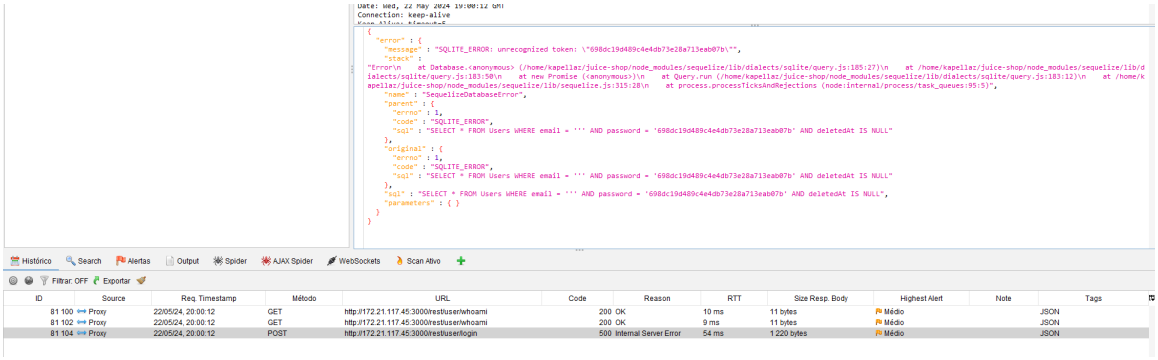


Figure 58: Error giving information about database

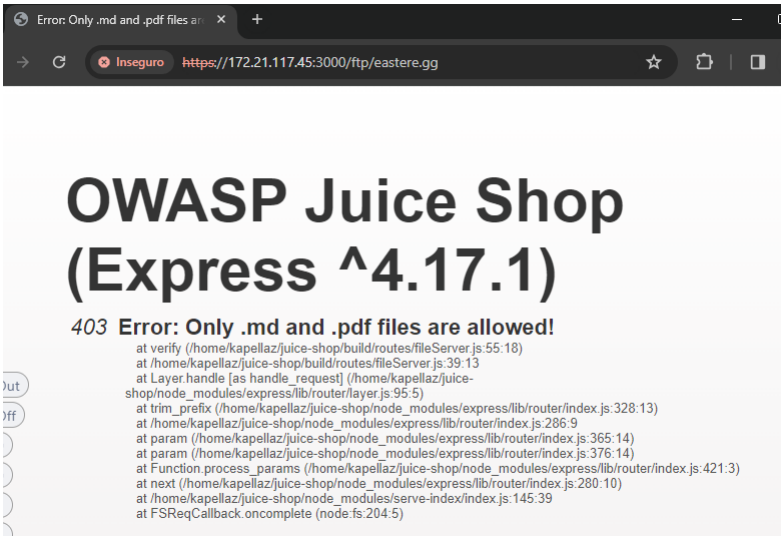


Figure 59: Stack Trace

4.13.2 WSTG-ERRH-02: Testing for Stack Traces

The guidelines state that this section was merged with the previous one, so we will not discuss this further.

4.14 Testing for Weak Cryptography

This involves testing the cryptographic features of the application, including encryption, decryption, hashing, and digital signatures. The objective is to identify flaws that could compromise the confidentiality, integrity, or authenticity of data or communications. The vulnerabilities stated in this topic are for applications that use HTTPS requests and responses. Since Juice Shop does not support HTTPS, this section is **not applicable**.

4.15 Business Logic Testing

This category focuses on testing the business logic of the application, including workflows, transactions, and calculations. The aim is to identify weaknesses that could allow bypassing or abusing the intended functionality or behavior of the application. From the information given by the project statement this section is **not applicable**.

4.16 Client-Side Testing

This section involves testing the client-side components of the application, such as JavaScript, HTML, CSS, and browser extensions. The goal is to find vulnerabilities that could allow modifying or compromising the user interface or user experience. As some of the topics in this section were already mentioned or are not applicable in this context we decided to cover the ones we think were relevant.

4.16.1 WSTG-CLNT-03: Testing for HTML Injection

WSTG-CLNT-03 aims to identify and evaluate HTML injection points in web applications. HTML injection vulnerabilities allow attackers to inject arbitrary HTML code into a web page, leading to various potential attacks, including session hijacking, content manipulation, and more. To test this, we tried to inject html in the search bar using this code:

```
<h1>Test HTML Injection</h1>
<img src="https://images.unsplash.com/photo-1595147389795-37094173bfd8?q=80&w=2069
&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D">
<br>
<a href="https://unsplash.com/">unsplash.com</a>
```

Figure 60: Html Injection

4.16.2 WSTG-CLNT-06: Testing for Client-side Resource Manipulation

Client-side resource manipulation vulnerabilities occur when an application accepts user-controlled input specifying the path of a resource (e.g., the source of an iframe, JavaScript, applet, or XMLHttpRequest). This vulnerability can enable an attacker to control URLs linking to resources on a webpage, potentially leading to XSS attacks or other malicious behavior. For example, if an attacker can control the src attribute of a script element, they can inject external JavaScript to execute in the victim's context. We can test this in Juice Shop by trying to change the cookies values using `<iframe src="javascript:document.cookie='token=teste';">` in the search area.

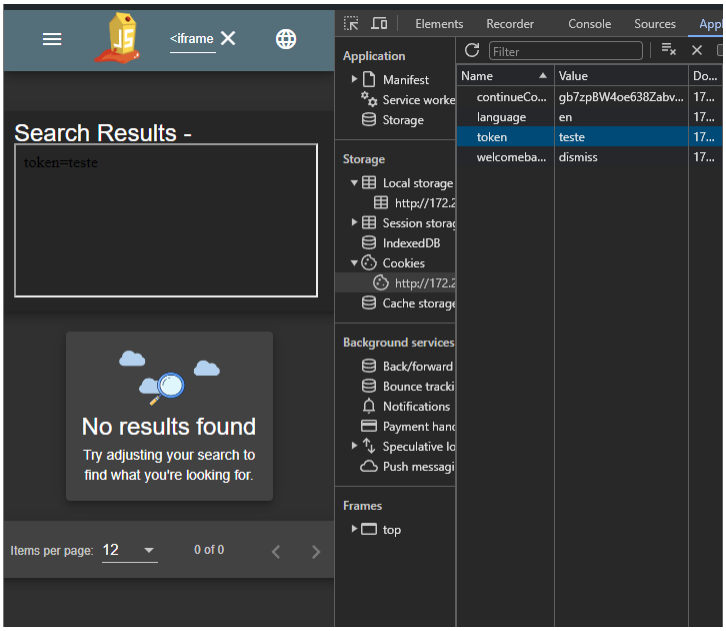


Figure 61: Changing token value

5 Web application security firewall

Building on our findings, we implement a Web Application Firewall (WAF) using ModSecurity to shield the Juice Shop from identified threats. The WAF's efficacy is evaluated by re-running our initial tests and observing its ability to detect and prevent attacks.

5.1 Scans

The differences between the automated scan and active scan were not so different with and without the application of the Web application security firewall so we did some manual testing in some cases to see if WAF made a difference. The ZAP reports are in the files sent with this report.

5.2 Fuzz attack

Here we tried again to access the admin credentials using SQL Injection and a fuzz attack. Without WAF, we successfully got the admin credentials with multiple SQL commands, however, using WAF not a single SQL command did pass.

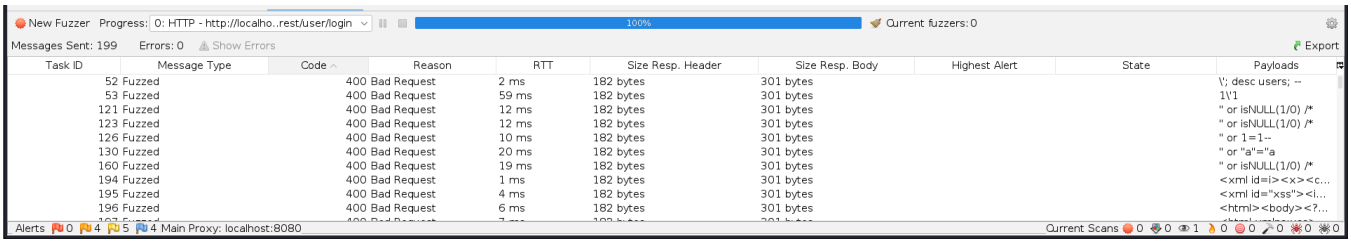


Figure 62: Fuzz attack

5.3 Manual Testing

Since the WAF does not cover all the WSTG points we will only cover some parts we find more relevant, starting with the SQL injection. As we saw before, it was possible to login in the site as an admin using some SQL strings like `' OR 1=1--`. Now it is not allowed.

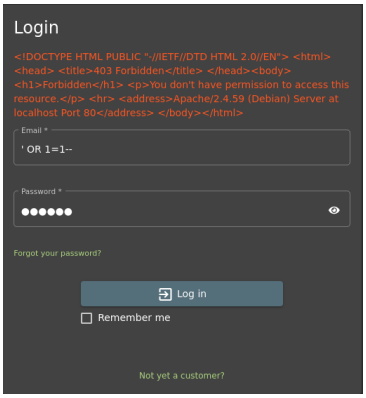


Figure 63: Login as admin with SQLi

It was also possible to use Poison Null Byte to access some files in `/ftp` and this is also blocked.

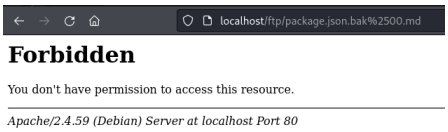


Figure 64: trying to download a file from the ftp endpoint

However, the ftp page should not be visible to any user, so we created a new rule to do that:

```
ecRule REQUEST_URI "@beginsWith /ftp" "id:100003,phase:1,deny,status:403,msg:'Access denied.'"
```

Now when we try to access `/ftp` we got the messages:

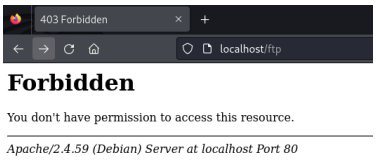


Figure 65: test `/ftp` with new rule

Figure 66: error.log file

After this we tried to test XSS and Html injection but they both still worked:

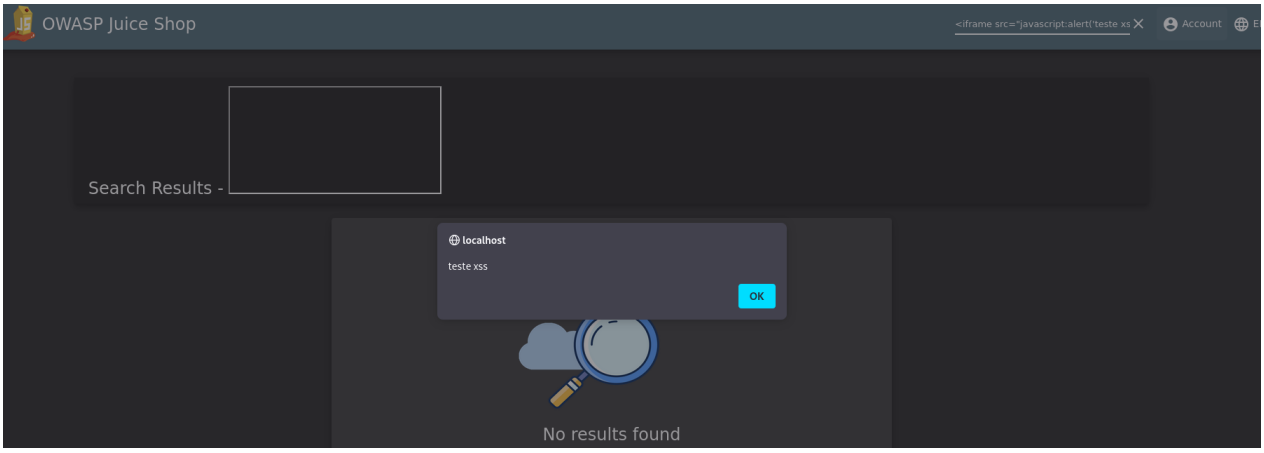


Figure 67: XSS injection

Figure 68: XSS injection

As we can see, WAF was able to detect and block some vulnerabilities, however, the rules we got from coreruleset were not enough to handle all the vulnerabilities.

6 Conclusion

This work provided the opportunity to analyze various vulnerable points in a web application called Juice Shop and test some of them using the guidelines from the Web Security Testing Guide (WSTG). Although the analysis was superficial, we recognize that with more detailed attention, we could uncover more vulnerabilities. Nonetheless, the overall objectives were achieved. The work also allowed for a better understanding of the functionality of OWASP ZAP. With it we conducted various types of scans, including automated scans, active scans and fuzzing attacks. We also performed manual tests that enabled us to discover more vulnerabilities and issues in Juice Shop following the methodology proposed by OWASP. Finally, we implemented a Web Application Firewall to try to mitigate the identified problems. Although the WAF increased security in some areas, it did not cover all vulnerabilities so we learned how to create rules for modsecurity and created and test one rule for ftp page access. In summary, this work not only achieved the initial objectives of identifying and testing vulnerabilities but also expanded our knowledge of security tools and practices, highlighting the importance of continuous and detailed analysis to ensure the security of web applications.

References

[1] Information Tecnology Security Slides

[2] Jorge Granjal, Segurança Prática em Sistemas e Redes com Linux, FCA, 2017

[3] OWASP Web Security Testing Guide, Retrieved May 19, 2024 from https://owasp.org/www-project-web-security-testing-guide/stable/

[4] Pwning OWASP Juice Shop, Retrieved May 19, 2024 from https://pwning.owasp-juice.shop/companion-guide/latest/

[5] ZAP Modes, Retrieved May 19,2024 from https://www.zaproxy.org/docs/desktop/start/features/modes/

[6] Download Bypass: Poison Null Byte, Retrieved May 22, 2024 from https://wiki.zacheller.dev/web-app-pentest/upload-download/error-only-.md-and-.pdf-files-are-allowed

[7] Securing Apache 2 With ModSecurity, Retrieved May 24, 2024 from https://www.linode.com/docs/guides/securing-apache2-with-modsecurity/

[8] How to Configure a simple reverse proxy for Apache2, Retrieved May 24, 2024 from https://dannyda.com/2021/12/07/how-to-configure-a-simple-reverse-proxy-for-apache2-on-linux-debian-ubuntu-kali-linux-etc/

[9] Modsecurity Core Rule Sets and Custom Rules, Retrieved May 25, 2024 from https://www.prosec-networks.com/en/blog/modsecurity-core-rule-sets-und-eigene-regeln/

[10] Making Rules, Retrieved May 25, 2024 from https://coreruleset.org/docs/rules/creating/