UNIVERSIDADE D
COIMBRA

Information Technology Security

Authors:
Edgar Duarte no. 2019216077
Tatiana Almeida, no. 2019219581

May 25, 2023

Contents

1 Introduction

This project has as main objective explore security vulnerabilities and threats of a web application, **Juice Shop**, being the final goal to build a firewall that protects the application from these attacks. In order to achieve this, we will use **OWASP ZAP** to perform automatic and active scans of the application as well as perform Fuzz attacks, and will use **modSecurity WAF** for the firewall implementation.

2 Architecture

2.1 Network structure

For web security testing, both the attacker and the web server will be hosted in the same machine, seeing that there was virtually no benefit in having them in different machines:

Figure 1: Scenario for security testing

Considering that all services(WAF and Juice Shop) are operating in isolated Docker containers, we utilize only one virtual machine for all configuration and testing.

2.2 Configurations

Juice Shop

Juice Shop can easily be run via a docker container. As such, it can be installed and run by doing the following commands:

```
Commands to install docker:

yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/
    centos/docker-ce.repo
yum install docker-ce docker-ce-cli containerd.io docker-buildx-
    plugin docker-compose-plugin

-----------------------------------------------------------------------


Commands to run Juice Shop in docker:
```

```
docker pull bkimminich/juice-shop
systemctl start docker
docker run -d -p 3000:3000 bkimminich/juice-shop
```

WAF

Just like Juice Shop, to use WAF we pulled a docker image and ran it:

```
docker run -d -p 1234:80 -e PARANOIA=3 -e BACKEND=http
    ://192.168.222.253:3000/ owasp/modsecurity-crs:apache
```

With this command, to use WAF, we will access the 192.168.22.253 address by the 1234 port. The PARANOIA parameter indicates how strict the rules imposed are being the minimum 1 and the maximum 4.

2.3 Automated scan

After completing the configurations, our next step was to conduct a automated scan.

Alert Name	Alert Level	Occurances
SQL Injection	**High**	1
Cloud Metadata Potentially Exposed	**High**	1
User Agent Fuzzer	**Informational**	108
Advanced SQL Injection	**High**	1
Backup File Disclose	**Medium**	31
Bypassing 403	**Medium**	5
CORS Header	**Medium**	145
Cookie Slack Detector	**Informational**	16

Some of the identified alerts were:

- **SQL injection** - this was one of the most concerning alerts we identified. This type of attack can destroy an application's database. An SQL injection is a method by which malicious actors can inject SQL commands into a SQL statement by way of web page inputs. These injected SQL commands can potentially alter and breach the security of a web application.
- **Cloud Metadata Potentially Exposed** - With a high level of importance, "cloud metadata potentially exposed" was another alert detected by the automated scan. Metadata contains other data about data, serving to describe documents, images or videos for easy retrieval, editing and archiving. Therefore, it is concerning when cloud metadata is exposed as this could result in a security risk. It is important to take measures to protect oneself from potential harm should this occur.

- **Backup file disclosure** - At a medium level of alertness, backing up files can pose a risk to web servers. These files could contain older or current versions of a file and may include sensitive data like password files or source code of applications. This type of issue can often lead to additional vulnerabilities or, worse, the disclosure of confidential information.

2.4 Active Scan

As a second type of testing, we perform an active scan where we modify the default policy to:

- **Threshold** - LOW
- **Strength**- INSANE

Despite numerous alerts that were generated, we have included in the table below only a few that are considered to be pertinent.

Alert Name	Alert Level	Occurances
Cloud Metadata Potentially Exposed	**High**	1
ELMAH Information Leak	**Medium**	1
Trace.axd Information Leak	**Informational**	2
.htaccess Information Leak	**Informational**	2
.env Information Leak	**Informational**	2
User Agent Fuzzer	**Informational**	123
Advanced SQL Injection	**High**	1
Source Code Disclosure - SVN	**Medium**	36
Backup File Disclose	**Medium**	31
Cookie Slack Detector	**Informational**	13

It is evident from the preceding tables that a considerable number of alerts were repeated in both automated and active scans. For example, SQL injection, Cloud Metadata Potentially Exposed, and Backup File Disclosure - alerts which have been previously explained - appeared in both tables.

2.5 Fuzz Attack

In this section we will perform a Fuzz attack to access an admin account by brute forcing our way in. Firstly we navigate to the login page and input a valid admin email (can be found in a post in the website) and type in a random password. We will be denied entry, as expected.

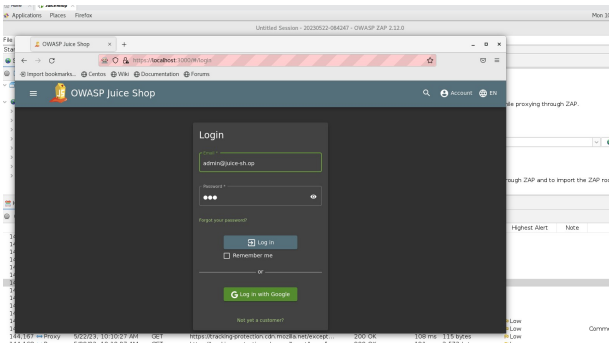

Figure 2: Initial setup

After that, we go to OWASP and find the request we just made. We analyze the contents of the package that was sent and double click on the password that was inputted ("123" in our case).

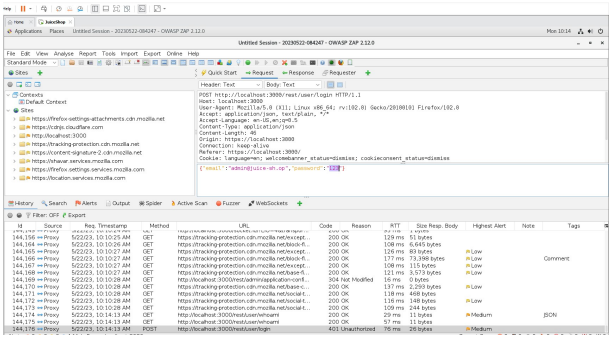

Figure 3: Request package

Next, we go to the Fuzz menu, to change the payload for the password variable, using a dataset of commonly used passwords.

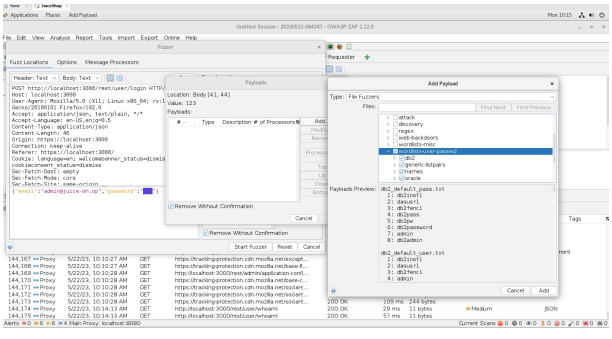

Figure 4: Setting payload

After running the Fuzz attack we can see that we successfully got the account's password (admin123).

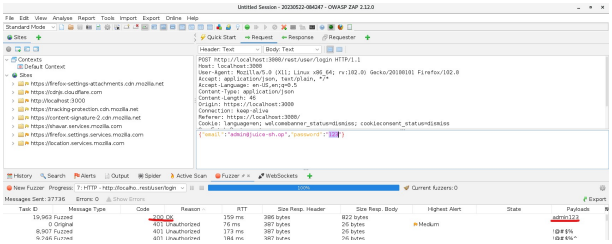

Figure 5: Password detected

More fuzz attacks will be used for further sections.

3 Web application security testing

3.1 Information Gathering

Conduct Search Engine Discovery Reconnaissance for Information Leakage

For search engines to operate properly, computer programs (also known as robots) continuously acquire data (called "crawling") from billions of web pages. These programs locate web content and features by following links from other pages or inspecting sitemaps. If a site uses a special file called robots.txt to list pages that should not be indexed by search engines, then those pages will be ignored.

In our application, since it is operating in a secured environment that is disconnected from the Internet, it is not feasible to take advantage of the existing search engines for "crawling" the "Juice Shop". Therefore, this topic does not apply to our project.

Fingerprint Web Server

Web server fingerprinting is a process of determining the type and version of web server used by a target. While these actions can be automated through testing tools, it is critical that researchers understand the basics behind how servers can be identified and why doing so is beneficial.

Having the ability to accurately identify what web server an application runs on enables security testers to decide if there is any potential for attack. Older versions of software that do not contain recent security updates may be particularly vulnerable to exploits due to their lack of protection from recently-discovered flaws.

In this instance, our intent was to obtain information about the web server hosting the "Juice Shop" application. We executed an HTTP request to the

application, which, even though it did not enable us to determine which server precisely, allowed us to ascertain important details such as the content-length.

```
[root@localhost ~]# telnet localhost 3000
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /HTTP/1.1
Host:localhost

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Sun, 21 May 2023 13:37:25 GMT
ETag: W/"7c3-1883e8776ac"
Content-Type: text/html; charset=UTF-8
Content-Length: 1987
Vary: Accept-Encoding
Date: Mon, 22 May 2023 17:42:29 GMT
Connection: close

<!--
  ~ Copyright (c) 2014-2023 Bjoern Kimminich & the OWASP Juice Shop contributors.
  ~ SPDX-License-Identifier: MIT
  --><!DOCTYPE html><html lang="en"><head>
  <meta charset="utf-8">
  <title>OWASP Juice Shop</title>
  <meta name="description" content="Probably the most modern and sophisticated insecure web application">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link id="favicon" rel="icon" type="image/x-icon" href="assets/public/favicon_js.ico">
  <link rel="stylesheet" type="text/css" href="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.css">
  <script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
  <script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
  <script>
    window.addEventListener("load", function(){
      window.cookieconsent.initialise({
        "palette": {
          "popup": { "background": "#546e7a", "text": "#ffffff" },
          "button": { "background": "#558b2f", "text": "#ffffff" }
        },
        "theme": "classic",
```

Figure 6: Response to the HTTP request made to the application

Review Webserver Metafiles for Information Leakage

Previously mentioned, web applications usually maintain a "robots.txt" file that lists the paths within the app that should not be indexed by search engines. However, if an attacker can access this file, they can easily find these areas and pinpoint their attack towards them as they likely contain sensitive information.

```
[root@localhost ~]# wget localhost:3000/robots.txt
--2023-05-22 14:05:38--  http://localhost:3000/robots.txt
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:3000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 28 [text/plain]
Saving to: 'robots.txt.1'

100%[=============================================================================================================================================>] 28          --.-K/s   in 0s

2023-05-22 14:05:38 (855 KB/s) - 'robots.txt.1' saved [28/28]

[root@localhost ~]# cat robots.txt
User-agent: *
Disallow: /ftp
```

Figure 7: Accessing "Robots.txt"

After analyzing this, we identified a region in the "/ftp" application that would not be indexed by search engines. Upon manually attempting to access this resource via the browser, we discovered that there is no form of protection for those who can gain entry, which poses a serious security risk.

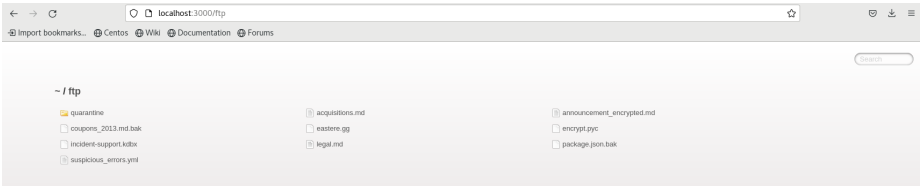

Figure 8: Accessing "/ftp"

Enumerate Applications on Webserver

A critical step in determining web application vulnerabilities is to identify the applications hosted on a web server. It is essential to not only identify this, but also provide a comprehensive list of all services running on the server, open ports, etc. This can give an attacker increased access to sensitive resources and consequently broaden their attack surface.

Performing an analysis with the command "nmap -Pn -sT -sV -p0-65535 localhost" can help to verify which ports are being used by the application.

```
[root@localhost ~]# nmap -Pn -sT -sV -p0-65535 localhost

Starting Nmap 6.40 ( http://nmap.org ) at 2023-05-24 07:07 PDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00066s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 65526 closed ports
PORT      STATE SERVICE     VERSION
22/tcp    open  ssh         OpenSSH 7.4 (protocol 2.0)
25/tcp    open  smtp        Postfix smtpd
111/tcp   open  rpcbind     2-4 (RPC #100000)
631/tcp   open  ipp         CUPS 1.6
3000/tcp  open  ppp?
8080/tcp  open  http-proxy?
10327/tcp open  unknown
36141/tcp open  unknown
40455/tcp open  tcpwrapped
40953/tcp open  unknown
```

Figure 9: List of ports being used

Review Webpage Content for Information Leakage

It is commonplace, even recommended, for programmers to provide comprehensive comments and metadata in their source code. Yet, any comments or metadata included in HTML code could uncover internal details that should not be accessible to potential malicious actors. To identify if any data is being exposed, reviews of the comments and metadata should be done. Moreover, certain applications may reveal information within redirect responses' content.

In this instance, when we examine the output of the ZAP alerts, we can observe some instances where ZAP recognized potential security vulnerabilities such as suspicious comments and likely data exposures.

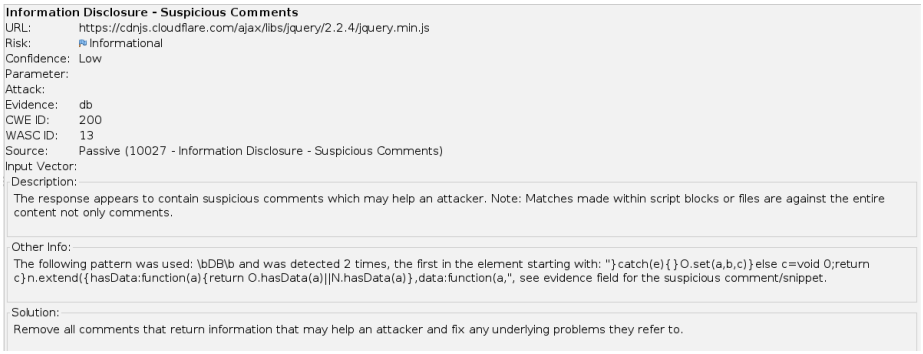

Figure 10: ZAP alert - Information Disclosure

Identify Application Entry Points

Identifying application entry points is a critical element of gathering information about the web application, as it can uncover vulnerabilities that can be leveraged to access restricted areas. In the case of Juice Shop we noticed that in some HTTP headers, session ids were being carelessly thrown around. This is a very bad security vulnerability.

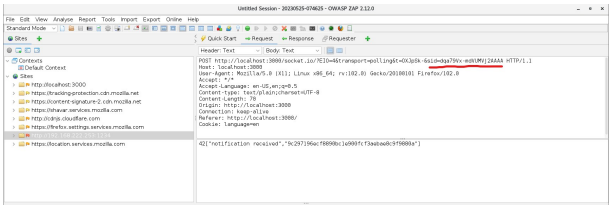

Figure 11: Exposed id

Fingerprint Web Application and Fingerprint Web Application Framework

This concept once more underscores the importance of "Fingerprinting" frameworks used by a web application in order to identify vulnerabilities that could potentially compromise the web application's security.

Figure 12: Framework and version

In this case, we can conclude that when attempting to open a file that is not in the .md or .pdf format, an error will be generated which reveals the framework and version utilized.

3.2 Configuration and Deployment Management Testing

Test Network Infrastructure Configuration

This test aims to review and analyze the system's network configurations in order to determine if they are or not vulnerable. It also has as an objective checking if the frameworks and systems used are secure from vulnerabilities that derive from unmaintained software or default settings and credentials. For Juice Shop, seeing that the deployment was made locally it is not easy to test and attack these network configurations. Even so, and as will be explored in the following sections and subsections, the application has a lot of misconfigurations and vulnerabilities. In fact, Juice Shop was created to be a playground for attackers to test their skills. All existing vulnerabilities can be verified in the 'metrics' endpoint, in which we can indeed verify that some network misconfigurations exist.

```
# HELP juiceshop_challenges_total Total number of challenges grouped by difficulty and category.
# TYPE juiceshop_challenges_total gauge
juiceshop_challenges_total{difficulty="3",category="XSS",app="juiceshop"} 2
juiceshop_challenges_total{difficulty="4",category="Sensitive Data Exposure",app="juiceshop"} 7
juiceshop_challenges_total{difficulty="3",category="Improper Input Validation",app="juiceshop"} 5
juiceshop_challenges_total{difficulty="2",category="Broken Access Control",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="6",category="Vulnerable Components",app="juiceshop"} 2
juiceshop_challenges_total{difficulty="3",category="Broken Authentication",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="5",category="Security through Obscurity",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="5",category="Insecure Deserialization",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="3",category="Broken Anti Automation",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="5",category="Broken Authentication",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="4",category="Injection",app="juiceshop"} 5
juiceshop_challenges_total{difficulty="4",category="XSS",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="1",category="Sensitive Data Exposure",app="juiceshop"} 2
juiceshop_challenges_total{difficulty="1",category="XSS",app="juiceshop"} 2
juiceshop_challenges_total{difficulty="3",category="Injection",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="2",category="Security Misconfiguration",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="4",category="Broken Access Control",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="5",category="Sensitive Data Exposure",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="1",category="Security Misconfiguration",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="4",category="Improper Input Validation",app="juiceshop"} 2
juiceshop_challenges_total{difficulty="5",category="Broken Anti Automation",app="juiceshop"} 2
juiceshop_challenges_total{difficulty="6",category="Cryptographic Issues",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="3",category="Broken Access Control",app="juiceshop"} 5
juiceshop_challenges_total{difficulty="5",category="Vulnerable Components",app="juiceshop"} 5
juiceshop_challenges_total{difficulty="4",category="Vulnerable Components",app="juiceshop"} 2
juiceshop_challenges_total{difficulty="2",category="Injection",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="3",category="Sensitive Data Exposure",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="4",category="Broken Authentication",app="juiceshop"} 2
juiceshop_challenges_total{difficulty="2",category="Sensitive Data Exposure",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="6",category="Security Misconfiguration",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="6",category="Broken Anti Automation",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="4",category="Cryptographic Issues",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="5",category="Injection",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="1",category="Unvalidated Redirects",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="2",category="Broken Authentication",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="1",category="Miscellaneous",app="juiceshop"} 4
juiceshop_challenges_total{difficulty="3",category="Security through Obscurity",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="2",category="XSS",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="1",category="Improper Input Validation",app="juiceshop"} 3
juiceshop_challenges_total{difficulty="6",category="Broken Access Control",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="6",category="Injection",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="2",category="Miscellaneous",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="4",category="Security through Obscurity",app="juiceshop"} 1
juiceshop_challenges_total{difficulty="6",category="Insecure Deserialization",app="juiceshop"} 1
```

Figure 13: All vulnerabilities

We do want to stress that we are aware that this endpoint is not a feature in any other site, meaning that if we go to, for example, Facebook and go to the 'metrics' endpoint, all of Facebook's vulnerabilities will not appear.

Test Application Platform Configuration

For this test, the aim is to ensure that default and know files have been removed, verifying that no debugging code or extensions were left from the testing phase of the application. Another objective of this test is to review the logging mechanisms set in place for the application.

For Juice Shop, we identified that some log files are exposed. To find these exposed logs, we used 'ffuf' a web fuzzer that allows for directory discovery.

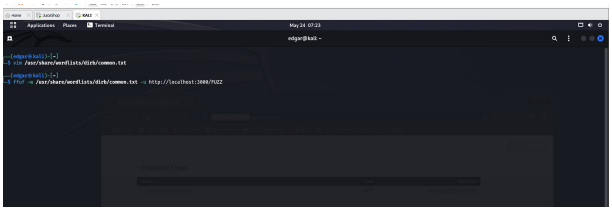

Figure 14: Use of ffuf

```
[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 5299ms]
    * FUZZ: rail

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 5292ms]
    * FUZZ: Rakefile

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 5107ms]
    * FUZZ: rank

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 5110ms]
    * FUZZ: ramon

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 5107ms]
    * FUZZ: random

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6219ms]
    * FUZZ: rar

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6220ms]
    * FUZZ: ranks

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6202ms]
    * FUZZ: rarticles

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6198ms]
    * FUZZ: rate

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6182ms]
    * FUZZ: ratecomment

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6166ms]
    * FUZZ: rateit

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6156ms]
    * FUZZ: ratepic

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6157ms]
    * FUZZ: rates

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6169ms]
    * FUZZ: ratethread

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6169ms]
    * FUZZ: rating0

[Status: 200, Size: 1987, Words: 207, Lines: 30, Duration: 6157ms]
    * FUZZ: ratings
```

Figure 15: Snippet of ffuf logs

The problem in using the above shown command is that Juice Shop website considers any string after 'localhost:3000/' as a valid request (response 200), redirecting to the main page when a matching url is not found. All of these false positives have the same length, 1987, meaning that if we filter these requests we will have less insignificant results.

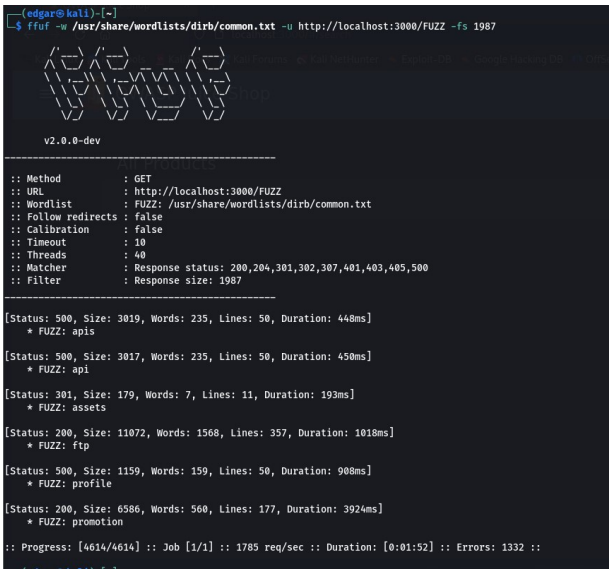

Figure 16: ffuf for localhost:3000/

Here we can see all the available urls paths. We will check some of them later, seeing that most of these contain vulnerabilities. For the logs, we dove deeper, and tried to search in the 'localhost:3000/support/' url:

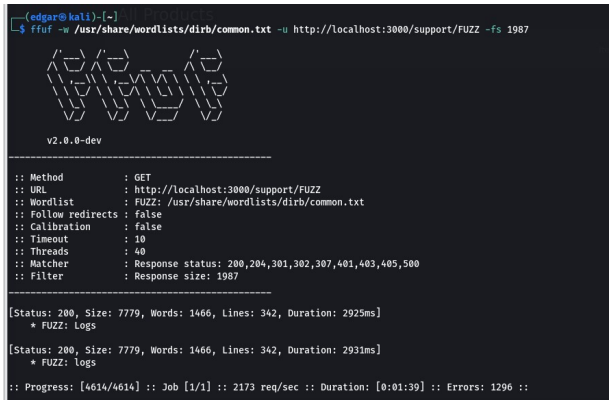

Figure 17: ffuf for localhost:3000/support

And just like that, we find two log paths. If we check them out we do indeed find an unprotected log file.

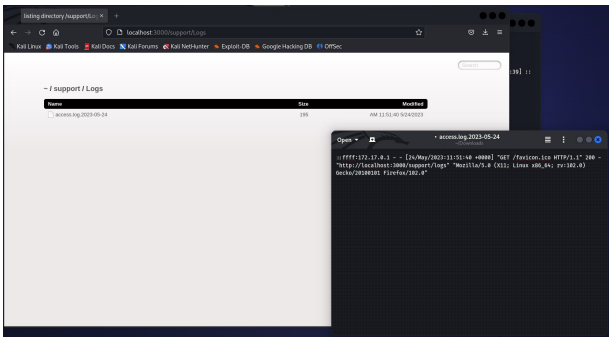

Figure 18: Log file

Test File Extensions Handling for Sensitive Information

In this test the objective is to verify if the type of technologies, languages and or plugins used are visible to the attacker. We might verify if the configurations are correctly set by sending a different file extension to a website and seeing if it is handled correctly. As such, using postman, we sent a POST request to the 'file-upload' endpoint:

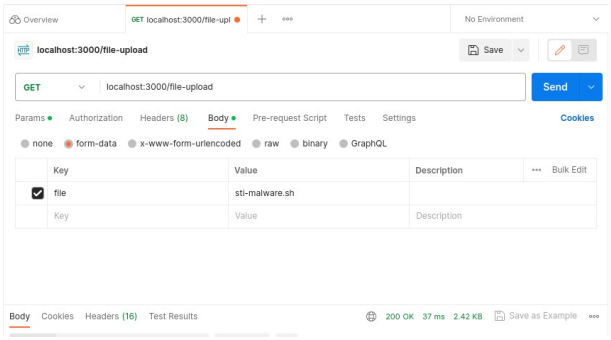

Figure 19: File upload via Postman

Seeing that the sent file, sti-malware.sh was accepted, it is clear that the application did not pass this test.

Review Old Backup and Unreferenced Files for Sensitive Information

This test is responsible for detecting backup and unreferenced files present in the live application. As was detected with ZAP, multiple of these files exist:

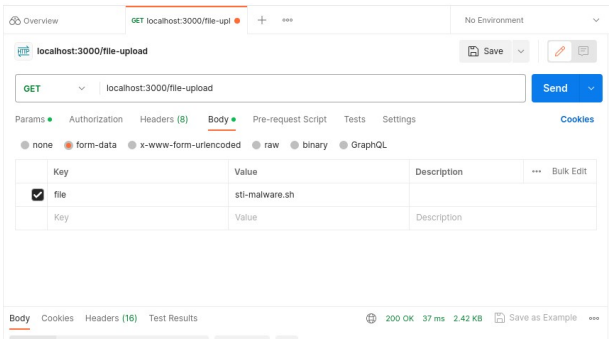

Figure 20: File upload via Postman

Enumerate Infrastructure and Application Admin Interfaces

The objective of this test is to identify hidden administrator interfaces and functionalities. No differences between interfaces were found in Juice Shop, making this test **not applicable**.

Test HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) uses a special header to let the website inform the browser which connections it should never establish. None of these headers were detected in Juice Shop, making this test **not applicable**.

Test RIA Cross Domain Policy

Rich Internet Applications (RIA) allow for controlled cross domain access using technologies like Oracle, Silverlight and Adobe Flash. This test's main objective is to review and validate the policy files. Seeing that Juice Shop does not use any of these technologies, this test is **not applicable**.

Test File Permission

This test's objective is to identify and review rogue files permissions. As was seen in a previous section, we have access to the '/ftp' endpoint which contains a number of unprotected confidential files. This is a grave security failure that should be addressed by the website.

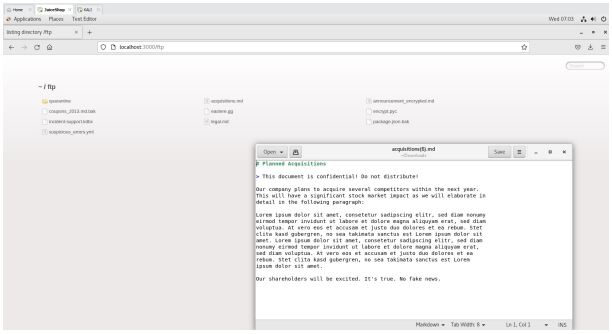

Figure 21: Unprotected confidential file

Test for Subdomain Takeover

This test is associated with DNS vulnerabilities, being it a forgotten or misconfigured domain. Seeing that the project is being hosted locally, DNS is not being used. As such, this test is **not applicable**.

Test Cloud Storage

This test verifies if the data saved in a cloud storage services can be viewed or tampered with. Seeing that Juice Shop does not offer or use such services, this test does **not apply**.

3.3 Identity Management Testing

Test Role Definitions

The idea of an app having roles for users, in order to group them into classes with different permissions, is a standard in web development nowadays. In Juice Shop, we found at least three roles, the Admin role, the Customer role and the Deluxe user, an acceptable approach. Even thought there are attacks that allow the Normal user to perform admin actions (will be explored bellow), we did not find an attack that allowed a Normal user to switch its account role to Admin.

Test User Registration Process

For this test, we shall verify if the user registration process is up to date with modern standards. As such, lets take a look at Juice Shop's user registration page:

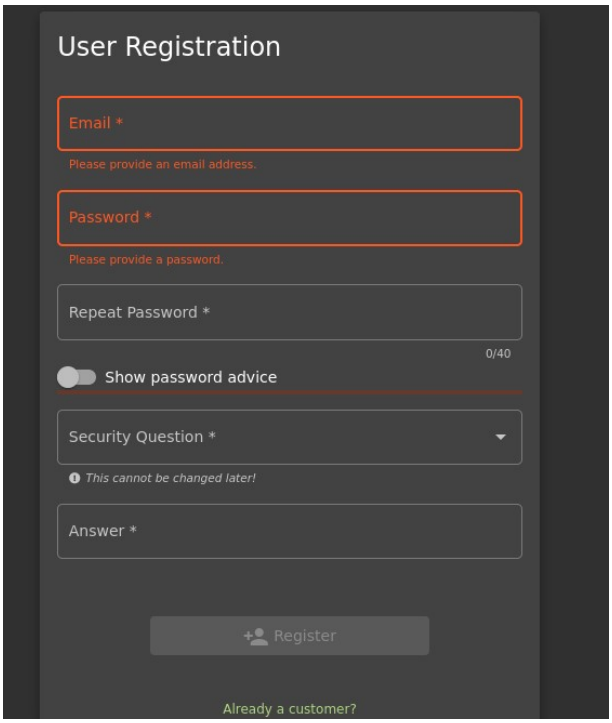

Figure 22: User registration page

The user registration requires an email, a password (5-40 characters long) that needs to be repeated and a security question. The user registration process should be very strict in order to allow the user to later login again to the same page but at the same time guarantee that the unwanted attackers access the account. As such, we consider that a minimum of 5 letters for a password is very weak. The website should require the user to pick a strong password (which contains both upper and lower case letters and at least one special symbol). It is also to note that the security question is a very outdated form of verifying one's identity and is prone to user forgetfulness. As such, we do not consider that Juice Shop's user registration mechanisms suffice modern days security threats.

Test Account Provisioning Process

In this test we need to validate that the registration of a new user on the website, which means, that we should verify if the information provided is indeed real (for example verify if the email is real). In the case of Juice Shop, the registration process both verifies the length of the password and if the email is syntactically correct, even though the existence of the email is never verified. In this sense, Juice Shop could improve its security measures.

Testing for Account Enumeration and Guessable User Account

For this test we pretend to see if it is possible to brute force our way into accounts by trying out combinations of user names and passwords. As was already seen with the use of Fuzzer, this is possible seeing that a user does not have a minimum amount of password attempts when trying to access an account. Also, when a user comments on a post, their email becomes visible, meaning that an attacker can copy that account's email and brute force their way into the account.

Testing for Weak or Unenforced Username Policy

This test tries to determine the structure of account names, their validity and predictability. Seeing that there are no usernames in Juice Shop, this test is **not applicable**.

3.4 Authentication Testing

Testing for Credentials Transported over an Encrypted Channel

At this point, we plan to evaluate how credentials are transmitted between client and server in an effort to determine whether the most commonly employed security measures are adequately protecting them from potential malicious actors that may be eavesdropping on the communication channel and attempting to intercept them (if sent in clear text). Specifically, this assessment will focus on verifying the use of HTTPS protocol. In regards to "Juice Shop", as it is designed to operate over HTTP and has no support for HTTPS, we have concluded that this criterion does not apply in our context.

Testing for Default Credentials

Many web applications and hardware devices come with default passwords for their built-in administrative account. Although some of these are randomly generated, they can often be static, which makes them easy targets for attackers to guess or access. Additionally, when new user accounts are created on these applications, the passwords may either be randomly generated by the application or manually set by personnel. Unfortunately, if these passwords do not meet security criteria, it can facilitate an attacker's ability to guess them correctly.

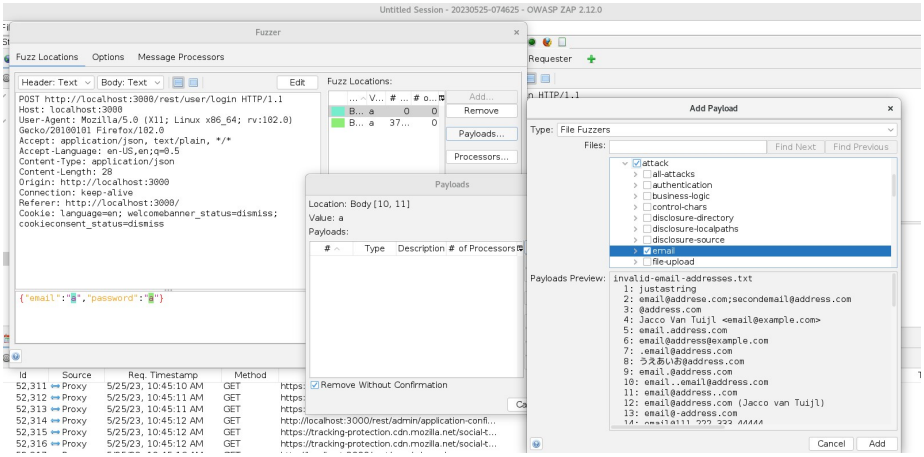

Figure 23: Setting Fuzzer for emails

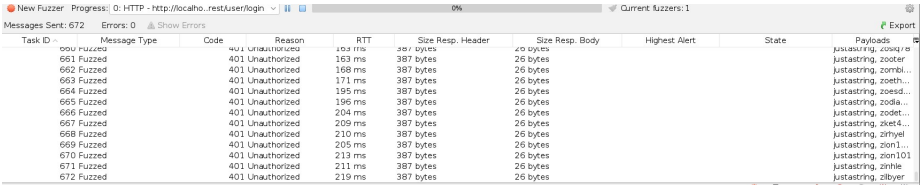

Figure 24: Fuzzer running

It is to note that this specific fuzzer test would probably take many hours, maybe even days to complete, seeing that every single email will be combined with every password, which takes a long time.

Testing for Weak Lock Out Mechanism

This test consists of seeing if the application is made to prevent fuzzing attacks. As was seen in previous sections, Juice Shop is clearly not prepared to deal with these kind of attacks.

Testing for Bypassing Authentication Schema

Testing the authentication schema requires understanding how the authentication process works and utilizing that knowledge to bypass the authentication mechanism.

At section 3.7, it is evident that the authentication process can be compromised via a SQL injection.

Moreover, we identified another vulnerability. Utilizing ZAP, if we make a successful login with any email address, we can acquire the session token of the same user. From there, we can add that session token to a query request,

granting us access to information that only becomes available when the user is logged in.

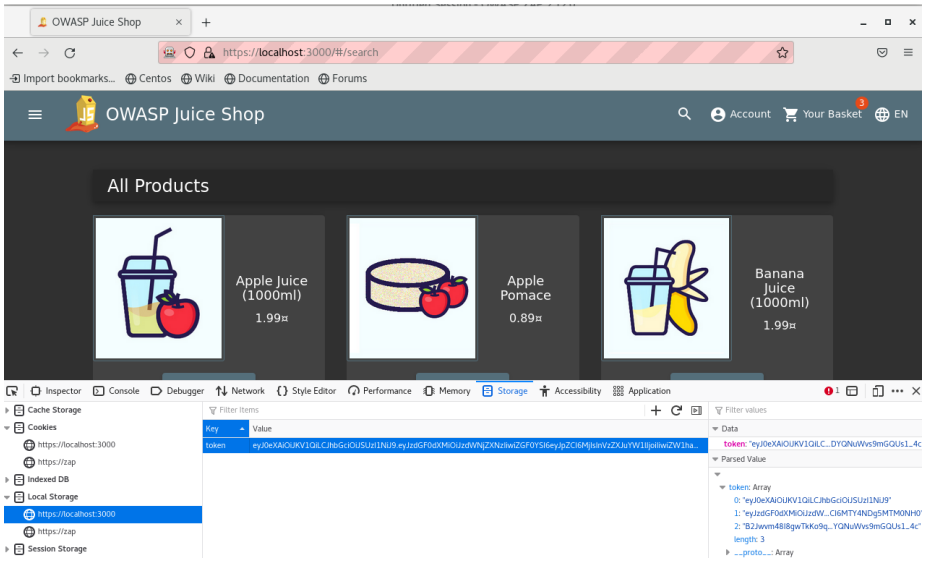

Figure 25: Session Token

Testing for Vulnerable Remember Password

The growing prevalence of username-password pairs has made it increasingly difficult for users to keep track of their credentials across multiple platforms. To facilitate the task, several solutions have been presented, such as applications with a "remember me" function that permits sustained authentication without requiring constant login information input; and password managers - including those integrated in browsers - that securely store and automatically enter data into forms with no user interaction.

We are now looking to evaluate the user's capability to access saved passwords locally, mainly through features such as "Remember Me".

Upon examining the "Juice Shop," it was confirmed that when clicking on the "Remember Me" button, information is stored in the local storage which could potentially lead to a breach of the user's account security. The following image illustrates this situation where the user's email address was retained even after closing and reopening the browser, thus amplifying fuzz attack risks for that user.

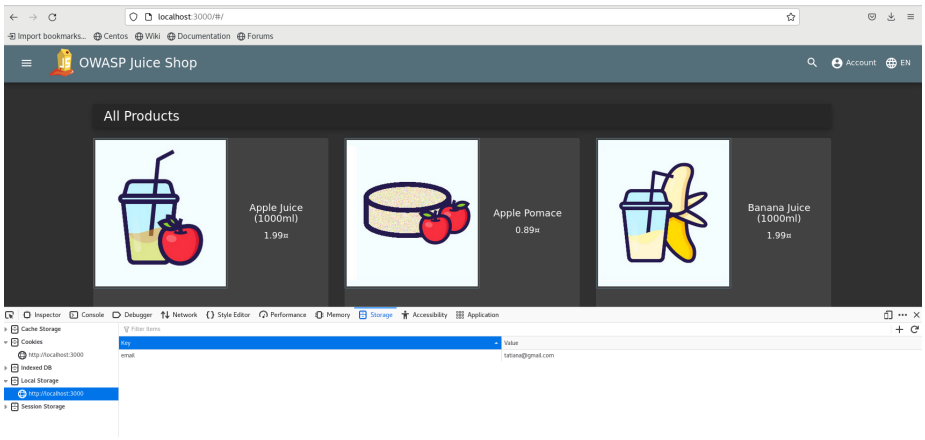

Figure 26: Retained email address

Testing for Browser Cache Weaknesses

It is essential to test if the application appropriately prevents browsers from retaining sensitive data. Browsers are able to save certain information for reference later and revisit websites by utilizing caching and history. If confidential information is displayed, it can be saved in the cache or history and become accessible when someone reviews the cache or clicks the Back button.

In the case of "juice shop", we could not locate any indications that the cache was retaining sensitive data.

Testing for Weak Password Policy

This evaluation is intended to assess the security policy of a given application in relation to user registration, focusing specifically on password restrictions. Poor quality passwords are a common target for attackers and thus it is essential for users to create strong passwords in order to protect themselves and their fellow users.

When attempting to register on the Juice Shop application, users are presented with a number of suggestions which are not mandatory. Passwords such as "12345" are allowed, as they still meet the chosen minimum length of five characters.

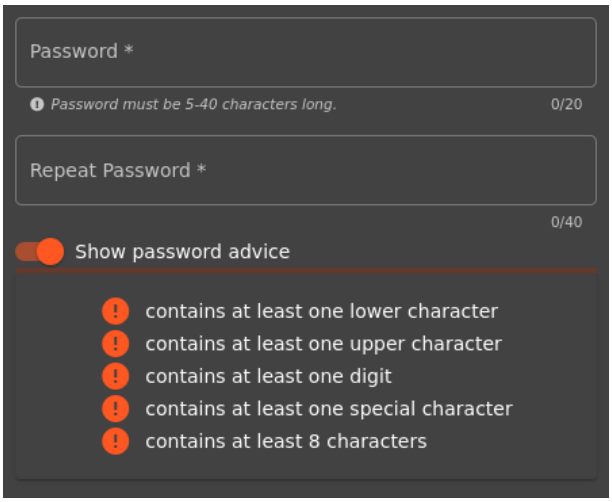

Figure 27: Password Advices

Testing for Weak Security Question Answer

Often referred to as "secret" questions and answers, security questions are commonly utilized to assist in the recovery of forgotten passwords.

When this type of questions is employed, the safest approach is to avoid basic or standard questions since they could be more vulnerable to fuzz attacks. This is precisely the issue that occurs in "juice shop" application. The type of questions are quite basic, as can be seen from the image bellow.

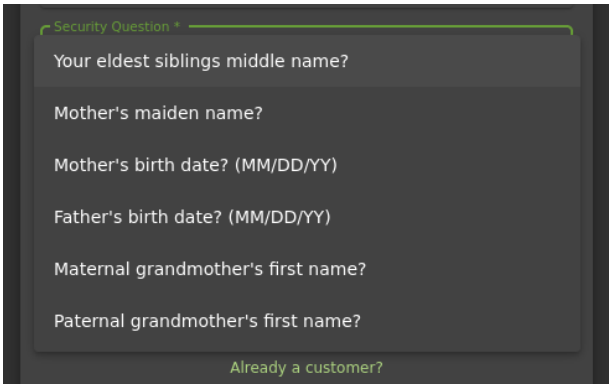

Figure 28: List of some security questions from "juice shop"

Testing for Weak Password Change or Reset Functionalities

For all applications that require a user account, which includes an email address and password, there must be a process in place to facilitate password

recovery.Generally, this process is accomplished by either sending an email or within the application itself.

In the case of the "juice shop", password recovery is conducted within the application, requiring only the answer to the security question and its associated email. However, this may prove problematic due to the fact that security questions are not always secure. It should also be noted that it is possible to reset a password to an identical one as previously used.

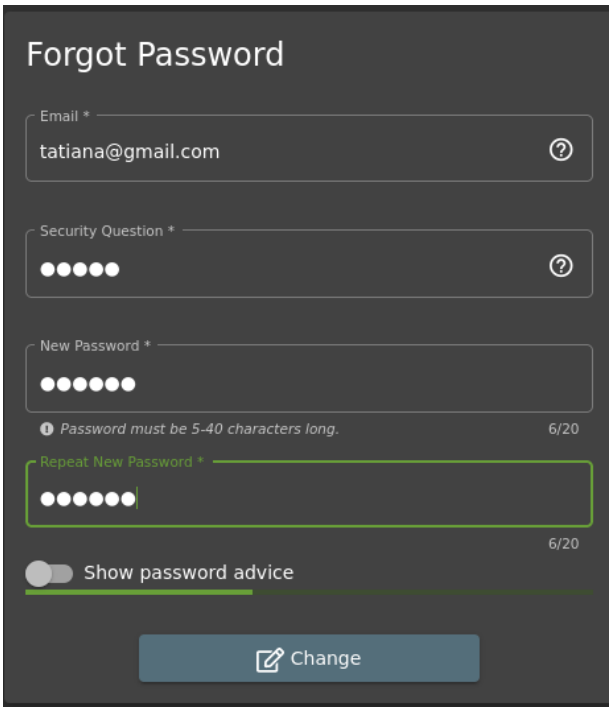

Figure 29: Forgot Password

Testing for Weaker Authentication in Alternative Channel

Even though the primary authentication mechanisms do not show any signs of vulnerabilities, alternative legitimate user channels for the same user accounts may still contain security flaws. These could potentially put the account's security at risk. Nonetheless, this is not applicable to the case of "juice shop" since there are no other authentication channels available.

3.5 Authorization Testing

Authorization tests are employed to validate whether or not we can gain access to resources which, in theory, we should not have authorization for—that is,

scenarios where it may be possible to circumvent security and obtain private resources.

Testing Directory Traversal File Include

We aim to further examine the robustness of the application in terms of providing users with navigational procedures that may give them unintentional access to files located on the server where it is hosted. In the case of the "Juice Shop", ZAP identified scenarios in which these strategies allowed for unauthorized access to these stored files.

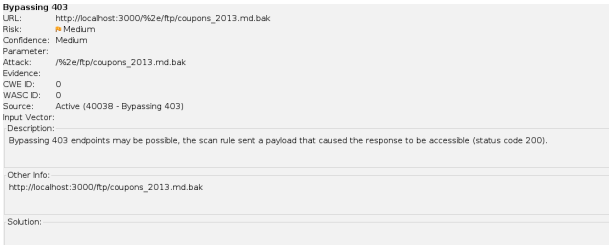

Figure 30: Situation detected by the ZAP, where the bypass occurred and where it was possible access files on the machine's filesystem

To test this vulnerability, we attempted to access the following address "http://localhost:3000/#/ftp/coupons_2013.md.bak".

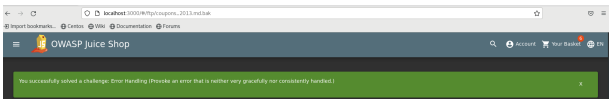

Figure 31

Testing for Bypassing Authorization Schema

This test focuses on verifying that the authorization schema is correctly implemented to allow roles and privileges access to restricted functions and resources.

Any vulnerability in the web application can be exploited by attackers, who can use modified identifiers to access other peoples' accounts and view their data, which is a type of attack called "horizontal escalation". To protect against such attacks, preventative measures must be taken.

In the case of the "juice shop," we identified a vulnerability of this type. By altering the bid value, it is possible to gain access to another user's shopping basket. To do so, simply inspect the webpage, select the "Storage" tab and then opt for "session storage" in the left corner, followed by clicking on the website link. Finally, modify the bid field with an alternative value.

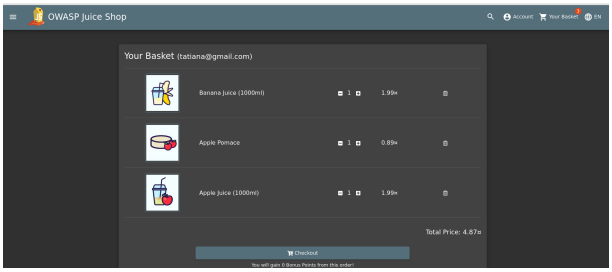

Figure 32: Shopping basket before changing bid

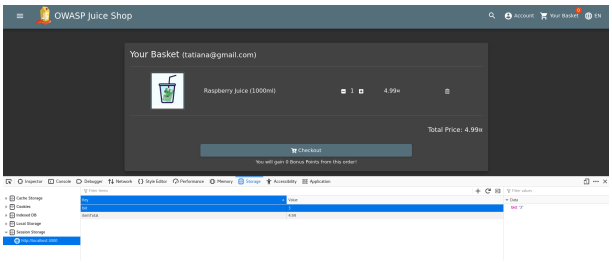

Figure 33: Shopping basket after changing bid

Testing for Privilege Escalation

During this phase, testers should verify that users cannot modify their privileges or roles inside the application in a manner that could lead to potential privilege escalation attacks.

For instance, we have conducted the test in *3.7 section* where a simple SQL injection allowed a normal user to gain access to an administrator account.

Testing for Insecure Direct Object References

Insecure Direct Object References (IDOR) is an issue that arises when applications offer direct access to objects based on user input. An attacker can exploit this security vulnerability to bypass authorization and gain direct access to resources such as database records and files within the system. IDOR can be caused by an application taking user-supplied input and using it to directly point to an object without performing proper authorization checks. This may give the attacker access to other users' database entries, files stored in the system, and more.

In the case of the "Juice Shop," we have already demonstrated its vulnerability via multiple SQL Injection attacks that enabled us to manipulate the "SQLite" database objects associated with the application.

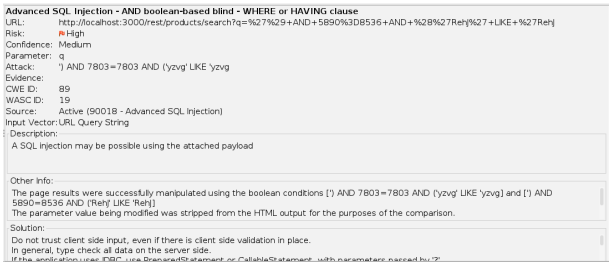

Figure 34: Advanced SQL Injection

3.6 Session Management Testing

Testing for Session Management Schema

To guarantee that users do not have to continually log in with every page change, websites implement various session management mechanisms, such as cookies. Cookies store a wide range of data about the user's identity, previously taken actions, preferences, etc.

Although we have not discovered any misuse of cookies, Zap has found some places where they are being utilized when unnecessary. An example of this can be seen in below image.

Figure 35: Cookie Slack Detector - ZAP

Testing for Cookies Attributes

As previously stated, the utilization of cookies is commonplace and their configuration must be properly done.

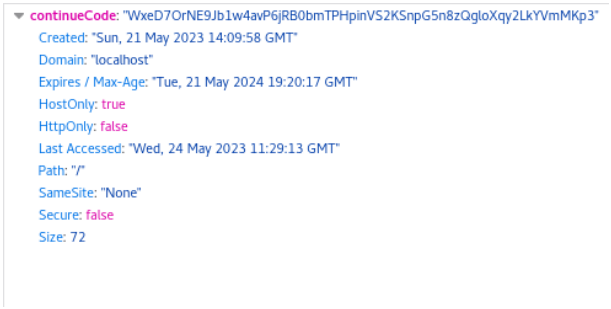

Figure 36: Cookies Attributes

Referencing the image above, we can see that the httpOnly variable is set to false. Incorporating the HttpOnly flag when creating a cookie helps reduce risks associated with client-side scripts accessing sensitive data in a cookie. This false attribute may lead to certain attacks such as Cross-Site Scripting.

Testing for Session Fixation

Session fixation occurs when the same session cookie value is maintained before and after authentication. This generally happens when session cookies are used to store information prior to login, such as for adding items to a shopping cart before processing payment.

In the context of the "Juice Shop", we were unable to identify any vulnerabilities of this type.

Testing for Exposed Session Variables

In order to protect against potential attacks, it is essential that data such as cookies, session IDs, and hidden fields are kept secure and out of public view. If someone external were to gain access to this type of information, the victim could be exposed to identity theft.

During our investigation with ZAP on the juice shop, we were able to find locations where the session ID was visible.

Figure 37: Session ID in URL Rewrite

Testing for Logout Functionality

This test serves to verify the effectiveness of the logout button. Through this test, we can conclude that the logout button is correctly implemented and that pressing it will clear the authentication token.

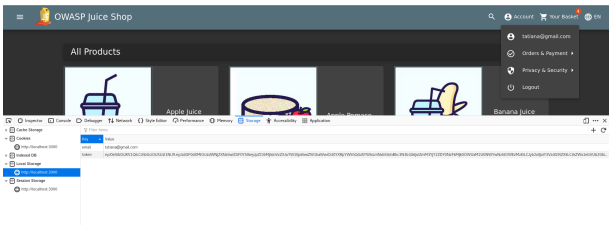

Figure 38: Logged

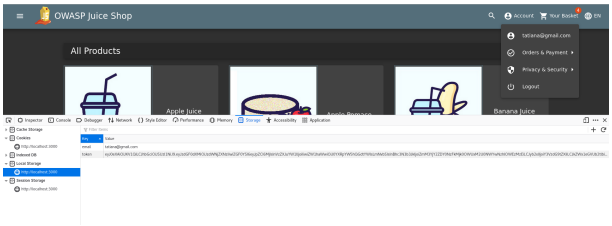

Figure 39: Logged out

Testing Session Timeout

"Session Timeout" is a functionality that, as its name suggests, disconnects the user from the web application after they have been inactive for a certain period of time.

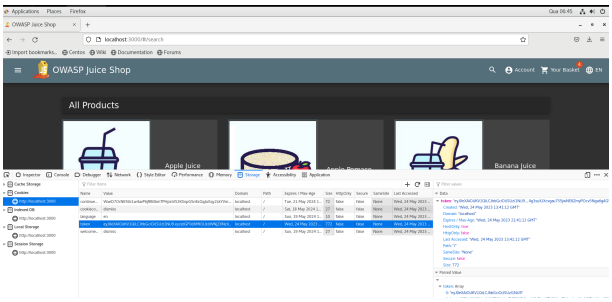

Figure 40: Session Timeout

From the image above, it is evident that the session timeout is 15 hours, which is not an acceptable length of time for an application's security.

Testing for Session Puzzling

This vulnerability arises when an application utilizes the same session variable for multiple purposes. Attackers can potentially exploit this weakness by accessing pages in a manner not initially anticipated by the developers, resulting in a situation where the session variable is set within one context and then utilized in another.

Since the Juice Shop utilizes JWT tokens that are configured with a mechanism to only validate tokens created with a secret key, we cannot verify this vulnerability.

3.7 Input Validation Testing

Testing for SQL Injection

An SQL injection test checks whether or not it is possible for users to send an unintended SQL query to the database with the objective of obtaining sensitive information or gaining access to previously blocked functionalities. We tested the resilience of the software to this type of attack in the login. First we found an admin email account in a product:

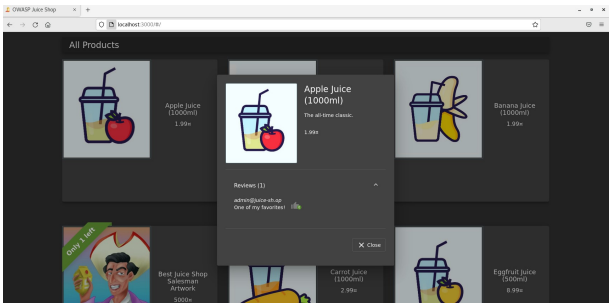

Figure 41: Admin email

Then we used that email as the email in the login field and added ' OR 1==1 - - to the field. This query makes the value of the statement always be True and ignores the following commands of the query. A random password was inserted in the password field.

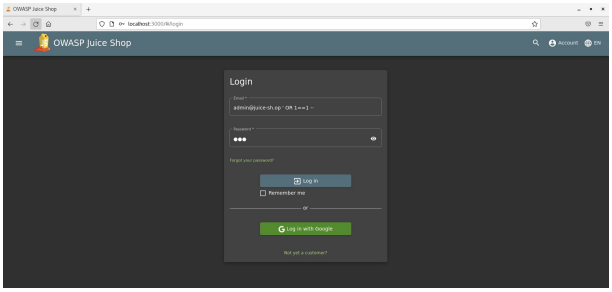

Figure 42: SQL Injection

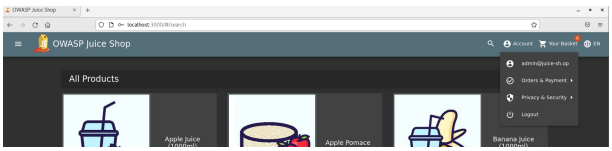

Figure 43: Access to admin account

As can be seen, we now have access to an admin account, allowing us to do heavy damage to the website. As such, the website is prone to SQL injection attacks.

3.8 Testing for Error Handling

Testing for Improper Error Handling and Testing for Stack Traces

The goal here is to ascertain whether the error messages generated by any type of application reveal any important information.

To demonstrate this concept, when looking for the "juice shop" example, by attempting to download a file that is not in .md or .pdf format, analyzing the resulting error message can enable us to understand which framework and its version are being utilized.

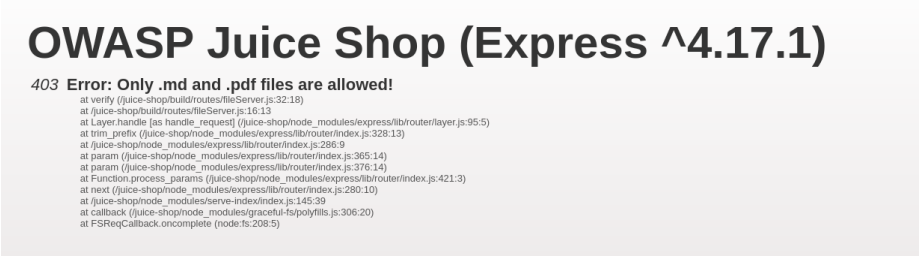

Figure 44: Framework and version

Developers often focus on the anticipated user experience, guiding users down what they term as "happy paths". However, this kind of thinking can

result in critical security issues, such as exposing internal object references, check versions, among other types of security problems. In order to address these security challenges, it is essential to take into account potential alternate paths users may take.

In the case of the "juice shop", one of the ways to exploit this vulnerability is, for example, to append "/rest" to the store's address. Through this action, we can observe an error with a stack-trace being displayed.

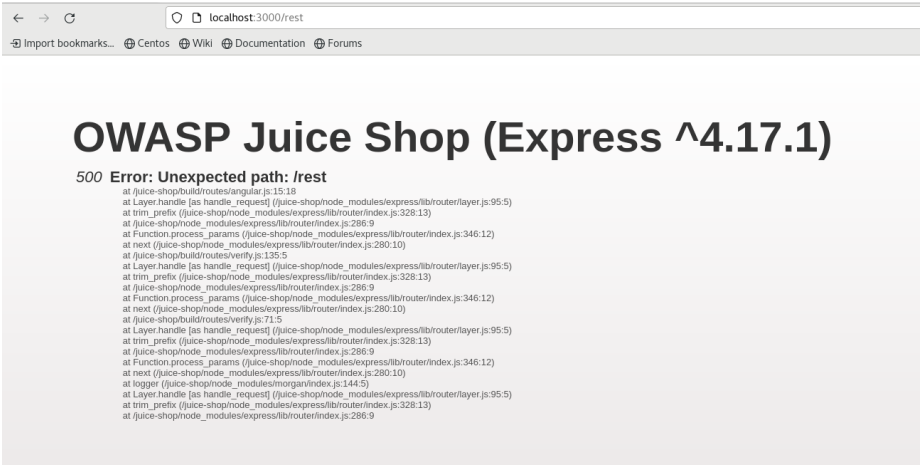

Figure 45: Stack trace

3.9 Testing for Weak Cryptography

Testing for DOM-Based Cross Site Scripting

3.10 Business Logic Testing

Business Logic Testing forces the attacker to think outside the box for vulnerabilities and attacks. As such, there is no specific scan or attack that we could do to detect these vulnerabilities. As such, we considered this section to be outside of the scope for the project.

3.11 Client Side Testing

Testing for DOM-Based Cross Site Scripting

DOM-based Cross Site Scripting (XSS Bugs) are a common security issue that is caused by the user's browser executing malicious code. This occurs when active browser-side content such as JavaScript takes data entered by a user and uses it in a way that results in the injection of hacker code. This injected code can then access all of the sensitive data stored within the application including passwords, login information, and credit card numbers.

In Juice shop, we can easily verify that it is not prepared to handle this type of attack. If in the search bar we put a simple HTML command (for example bold text) we can verify that the inputed text comes out as bold. This means that there is no protection against user inputed data.

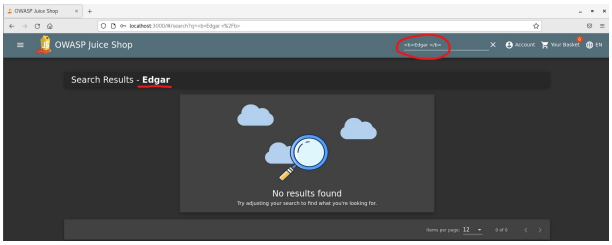

Figure 46: HTML code from user input displayed

With this knowledge, more interesting results can now be performed such as injection JavaScript scripts or harmful payloads. For example, we can introduce the following payload that displays the InforEstudante site:

```
<iframe width="100%" height="166" scrolling="no" frameborder="no"
    allow="autoplay" src="https://inforestudante.uc.pt/nonio/
    security/login.do"></iframe>
```

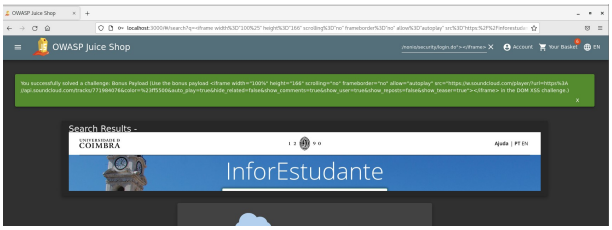

Figure 47: Displaying InforEstudante in the Juice Shop

As can be seen, the application is prone to this kind of attacks.

Testing HTML, CSS and JavaScript injections

As was seen before, HTML injections are possible. To verify is CSS and JavaScript injections are possible further tests were conducted. To test CSS injection, a paragraph with style red was inputted:

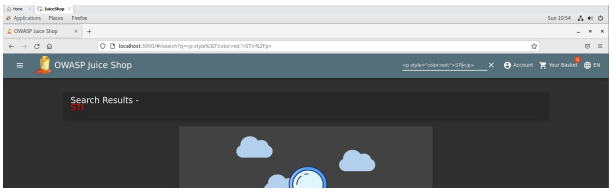

Figure 48: Displaying a red Paragraph

To test JavaScript injections, we inputted the following code in the search bar:

```
<a href="javascript:console.log('javascript');alert('javascript')">
    Link</a>
```

This code will create a clickable hyperlink that when clicked shows the following alert:

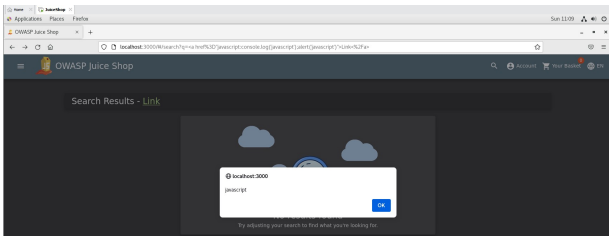

Figure 49: JavaScript injection

With this we can conclude that the application is not protected against HTML, CSS and JavaScript injections.

Testing for Client-side URL Redirect

The website contains an allowed list of URLS to which the user might be redirected to. This means that if we attempt to redirect to a website that is not in that allowed list, the website will sound an alert and deny the redirection:

```
http://localhost:3000/redirect?to=https://inforestudante.uc.pt/
    nonio/security/login.do
```

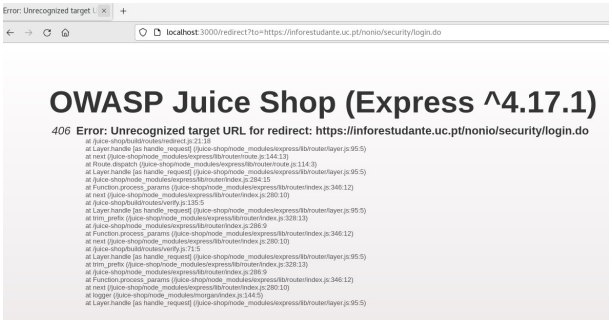

Figure 50: Attempt at URL redirect

On the other hand, the website allows for redirect to, for example, the GitHub of the website (https://github.com/bkimminich/juice-shop). With this knowledge, if we add a parameter to the initial URL with an allowed redirect URL, we might successfully redirect the user to the wrong address. As such, the URL used for the attack was as follows:

```
http://localhost:3000/redirect?to=https://inforestudante.uc.pt/
    nonio/security/login.do?https://github.com/bkimminich/juice-
    shop
```

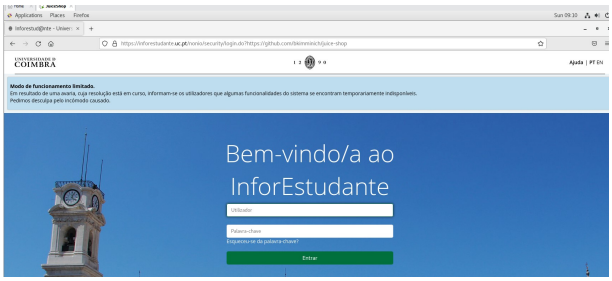

Figure 51: Successful Redirect

Testing for Client-side Resource Manipulation

Similar to the before mentioned attacks, a client-side resource manipulation attack involves the ability to control the URL that links to some content. When this vulnerability is present, an attacker might be able to interfere with the expected application behaviour by rendering undesired content.

In order to verify if Juice Shop is resistant to this type of attack, we inputted the following URL:

```
localhost:3000/#/search?q=<iframe src="javascript:alert(document.
    cookie)">
```

The idea behind this URL is for the page to open the page and show an alert containing information about the document's cookies. The result was as follows:

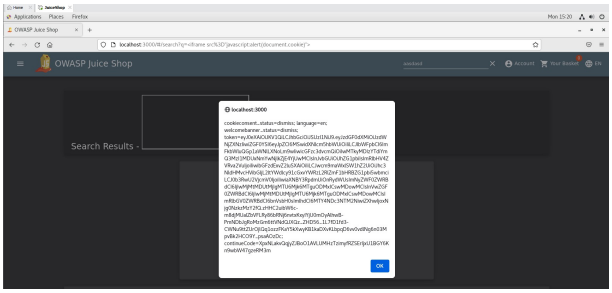

Figure 52: Successful Redirect

As can be seen, the alert was displayed to the user, meaning that Juice Shop failed in preventing this type of attack.

Testing Cross Origin Resource Sharing

Cross-origin requests require an origin header that identifies the domain that initialized the request which is sent to the server, using a protocol defined by CORS, for example HTTP headers. An Cross Origin Resource Sharing attack consists in an attacker intercepting these HTTP headers, revealing how CORS is used, especially which domains are allowed. To test this, we need to verify the origin header to see what domains are allowed. As such, using a ZAP manual scan, we intercepted the response to a POST package with the following contents:

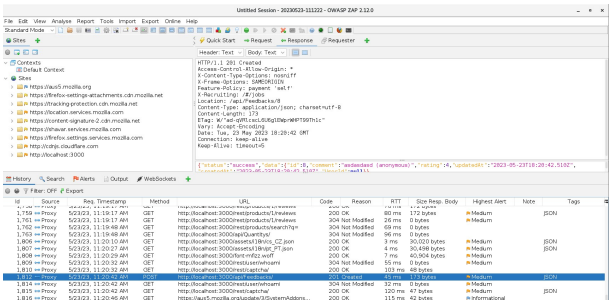

Figure 53: HTTP response header that shows which domains are valid

As such, the CORS configurations are exposed, which could server as useful information for attacks.

Testing for Cross Site Flashing

Cross-site Flashing (XSF) is similar to DOM-based Cross Site (XSS) in the sense that they both can be performaed by loading undesired or unintended objects/files to a user's page. XSF, as is suggested in the name, is the result of the loading of an evil Flash File. Seeing that Juice Shop does not use Flash, this test is **not applicable**.

Testing for Clickjacking

Clickjacking is the technique that maliciously manipulates a user's website in order to induce them to click on content which performs actions to which the user is unaware.

To achieve this, in the search bar we inputted the following iframe that displays a fake JuiceShop to the user:

```
<iframe src='fictitious.html'>
```

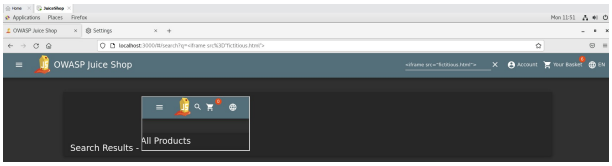

Figure 54: Successful ClickJacking

As can be seen, the page successfully loaded an interactive fake website, which means that the application is not protected against this type of attack.

Testing WebSockets

It is the server's responsibility to make sure that the Origin header is valid in the WebSocket handshake. If this step is not verified, the server may accept connections from any origin which could lead to attackers communicating with the WebSocket server cross-domain. To check this, using OWASP we checked if we could replay previous WebSocket requests:

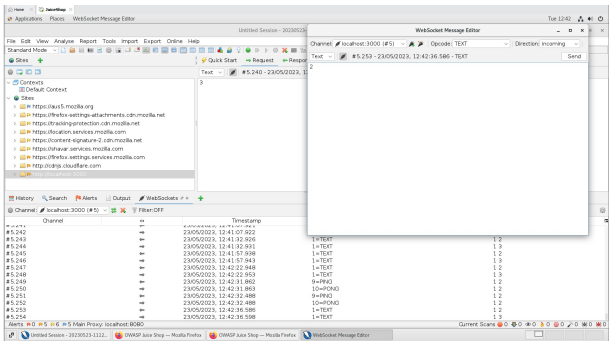

Figure 55: Repeated Websocket requests

We verified that the packages were replayed successfully even when opening a new connection, which could mean that the server might not be checking the WebSocket's handshake origin header, making the website prone to this type of attacks.

4 Web application security firewall

With a PARANOIA level of 3, when applying an automatic scan to the entire website using threshold low and attack strength at insane the results were as follows:

Alert Name	Alert Level	Occurances
Cloud Metadata Potentially Exposed	High	1
User Agent Fuzzer	Informational	61
Source Code Disclosure - File Inclusion	High	1
CORS Header	Medium	37
Insecure HTTP Method	Informational	378
Cookie Slack Detector	Informational	14

As can be observed, there are a lot less alerts than before, but there are some new informational alerts like insecure HTTP methods. As can be seen, for example, the SQL Injections supposedly have been resolved. As such, we decided to manually verify this:

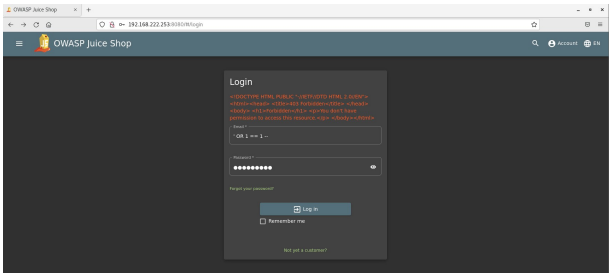

Figure 56: SQL Injection Prevented

We also manually checked for other previously explored vulnerabilities to verify if they are still possible. First we saw if the /ftp and /support/logs endpoints are still available:

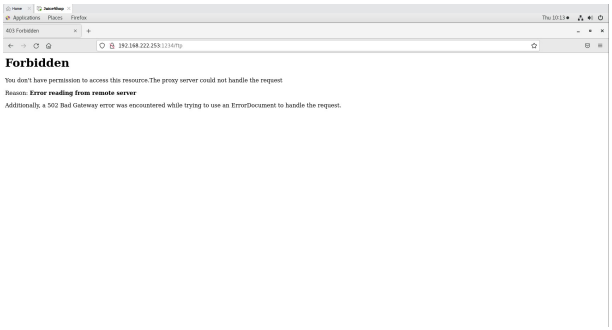

Figure 57: FTP page not authorized

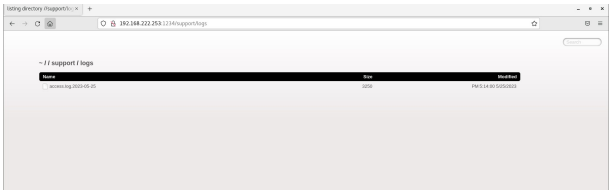

Figure 58: Logs still visible

As can be seen, even thought the ftp endpoint was successfully blocked, the support/logs endpoint is still available. Other tests we conducted was to see if JavaScript injections and clickjacking were or not prevented:

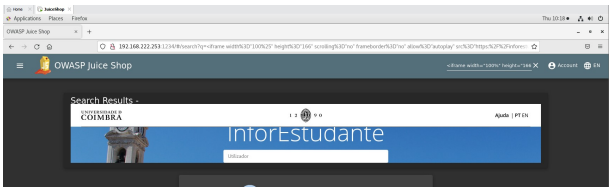

Figure 59: Java Injection still possible

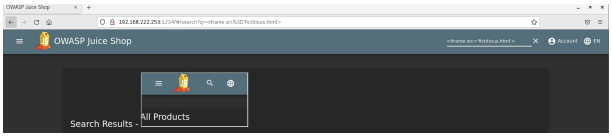

Figure 60: Clickjacking still possible

As can be seen, the current implementation of WAF does not prevent these attacks. We could increase the Paranoia level to level 4, but this makes the application almost unusable (we couldn't even login with the administrator valid password).

The introduction of WAF helped Juice Shop greatly in what comes to protecting against some big vulnerabilities like SQL Injections. Even so, it did not suffice to prevent all the vulnerabilities. As such, more and stricter rules might be needed to prevent more vulnerabilities.

5 Conclusions

In this work, we utilized the OWASP Juice Shop as a platform to investigate web application security. Taking advantage of ZAP and Kali, we conducted comprehensive and detailed tests that provided us with an understanding of the existing vulnerabilities within the application.

Implementing the firewall was an important stage in ensuring the security of the Juice Shop. Although some vulnerabilities still remain, their number has been reduced.

Bibliography

[1] Fuzz faster u fool. https://github.com/ffuf/ffuf. Accessed:10-05-2023.

[2] Fuzzing. https://owasp.org/www-community/Fuzzing. Accessed:10-05-2023.

[3] Owasp web security testing guide. https://owasp.org/www-project-web-security-testing-guide/. Accessed:10-05-2023.

[4] Some challenge solutions. https://pwning.owasp-juice.shop/appendix/solutions.html. Accessed:10-05-2023.

[5] Web security testing guide - v4.2. https://owasp.org/www-project-web-security-testing-guide/v42/. Accessed:10-05-2023.

[6] Jorge Granjal. *Segurança Prática em Sistemas e Redes com Linux*. FCA, 2017.