



UNIVERSIDADE D  
**COIMBRA**

**Relatório do Assignment 3**

**Segurança em Tecnologias da Informação**

Bruno José Silvério da Silva 2021232021

Diogo Emanuel Matos Honório 2021232043

# 1. Introdução

Este relatório foi elaborado no âmbito da cadeira de Segurança em Tecnologias da Informação, com o objetivo de aplicar e validar técnicas de teste de segurança em aplicações web, seguindo as metodologias propostas pelo [OWASP Web Security Testing Guide](#) (WSTG).

Através de um ambiente controlado, construído com recurso a máquinas virtuais, foi simulado um cenário realista onde uma aplicação vulnerável, o OWASP Juice Shop, é submetida a diversas análises de segurança, sem Firewall implementada.

A implementação do WAF, Web Application Firewall, foi bem-sucedida, no entanto não foram realizados testes com o mesmo.

A ferramenta principal utilizada para conduzir os testes foi o OWASP ZAP (Zed Attack Proxy), que permitiu executar ataques automáticos e personalizados, explorar endpoints manualmente e aplicar técnicas como fuzzing, SQLi e análise de cabeçalhos HTTP.

A estrutura do relatório segue a estrutura proposta pelo enunciado com adição de uma secção de anexos onde se encontram alguns dos resultados obtidos e redireccionamento para os ficheiros csv resultantes.

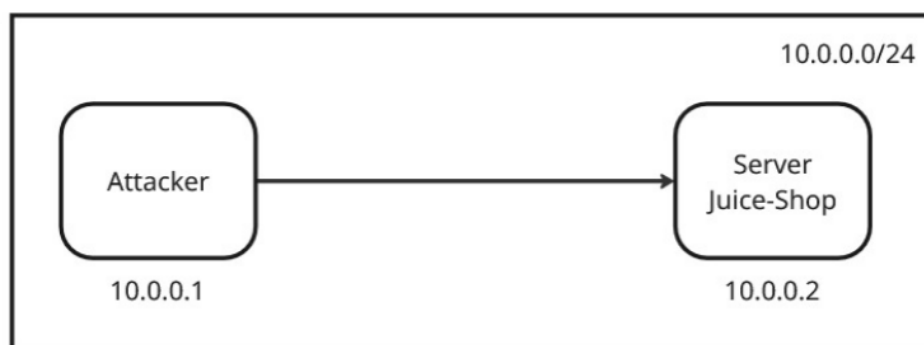
## 2. Arquitetura

### 2.1. Estrutura da Rede

A realização do trabalho prático foi desenvolvida com recurso ao uso de máquinas virtuais que representam cada um dos papéis principais para a resolução do mesmo.

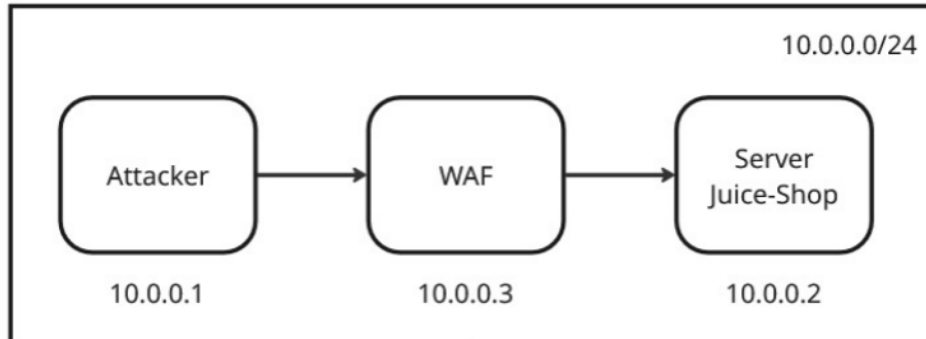
#### Cenário sem Firewall

No cenário sem a implementação de *firewall* apenas são usadas duas máquinas virtuais, uma com o atacante e outra com a aplicação web, como demonstrado na figura abaixo.



## Cenário com Firewall

Em relação ao cenário com a *firewall* implementada, ou seja, o WAF, é usada outra máquina que executa o *Apache 2* com *ModSecurity* que serve como monitorização, filtro e bloqueio de tráfego HTTP para a aplicação web.



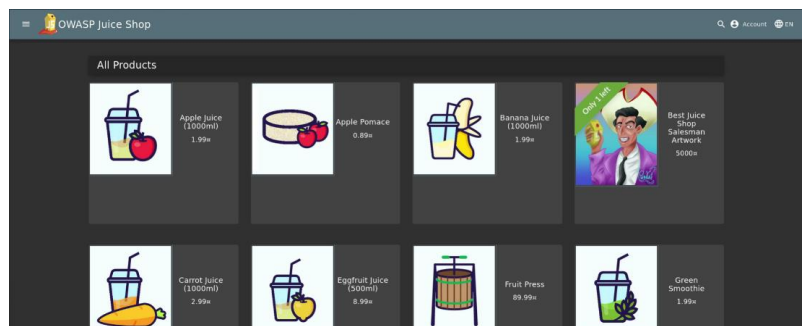
## 2.2. Serviços

### OWASP JUICE SHOP

O OWASP Juice Shop é uma aplicação web desenvolvida com o objetivo de simular um ambiente real suscetível a diversos tipos de ataques de segurança, sendo amplamente utilizada para fins educativos e de testes.

Neste projeto, a aplicação foi executada na máquina virtual “Server Juice-Shop”, atribuída com o endereço IP 10.0.0.2, recorrendo à utilização do Docker como plataforma de virtualização e contenção. A figura abaixo ilustra o processo de execução do serviço através de um container Docker e o respetivo *website* da aplicação.

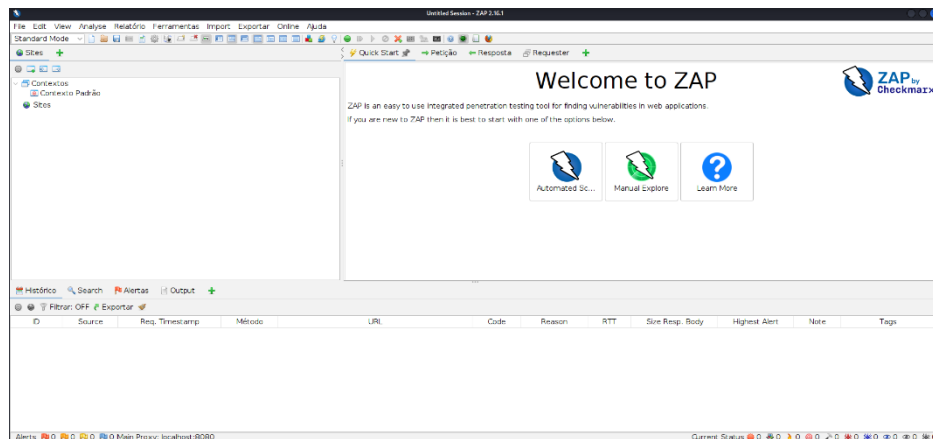
```
(admin_sti@admin)-[~]
$ sudo docker run --rm -p 10.0.0.2:3000:3000 bkimminich/juice-shop
[sudo] senha para admin_sti:
info: Detected Node.js version v20.19.0 (OK)
info: Detected OS linux (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 19 of 19 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file index.html is present (OK)
info: Required file main.js is present (OK)
info: Required file tutorial.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file vendor.js is present (OK)
info: Required file styles.css is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Server listening on port 3000
```



## OWASP ZAP

De modo a testar a aplicação web *Juice Shop*, foi utilizada a ferramenta OWASP ZAP (*Zed Attack Proxy*). O ZAP permite a realização de diversos tipos de testes, como *passive scanning*, *active scanning*, *fuzzing*, ..., e ataques personalizados, sendo especialmente eficaz no apoio à análise de segurança automatizada e manual.

No contexto deste projeto, a ferramenta foi fundamental para identificar falhas de segurança, simular cenários de ataque e avaliar o comportamento da aplicação sob os mesmos.



## WAF

Um WAF (Web Application Firewall) é uma solução de segurança que atua entre o utilizador e a aplicação web com o objetivo de monitorizar, filtrar e bloquear tráfego HTTP indesejado. Diferente dos firewalls tradicionais, que operam ao nível da rede, o WAF foca-se especificamente no tráfego da camada de aplicação (HTTP/HTTPS), sendo uma defesa eficaz contra ataques como SQL Injection, Cross-Site Scripting (XSS), Remote File Inclusion, entre outros.

No contexto deste projeto, foi utilizado o ModSecurity como WAF, integrado num servidor Apache2, conforme a configuração demonstrada na figura abaixo.

```
(admin_sti@admin)-[~]
$ sudo apt install apache2 libapache2-mod-security2 -y
apache2 já é a versão mais nova (2.4.63-1).
libapache2-mod-security2 já é a versão mais nova (2.9.8-1.1).
Os seguintes pacotes foram instalados automaticamente e já não são necessários:
  icu-devtools  libglapi-mesa  liblbfgsb0  libpython3.12-minimal  libpython3.12t64  python3.12-tk
  libgeos3.13.0  libicu-dev  libpoppler145  libpython3.12-stdlib  python3-requests-ntlm  strongswan
Utilize 'sudo apt autoremove' para os remover.

Resumo:
  Atualizando: 0, Instalando: 0, Removendo: 0, Não atualizando: 596
```

Após a instalação do Apache2 e do ModSecurity, procedeu-se à alteração do seu ficheiro de configuração, de forma a que o módulo deixasse de operar em modo apenas de deteção (Detection Only) e passasse a atuar em modo de prevenção ativa, ou seja, a bloquear pedidos HTTP de acordo com as regras definidas.

```
GNU nano 8.4 modsecurity.conf-recommended *
# -- Rule engine initialization
#
# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine On
```

Para reforçar a proteção, foi também integrado o OWASP Core Rule Set (CRS), um conjunto abrangente de regras mantido pela comunidade OWASP, que cobre as principais vulnerabilidades conhecidas. Estas regras adicionais foram incorporadas na configuração do ModSecurity para permitir uma análise mais rigorosa do tráfego HTTP e bloquear automaticamente padrões de ataque reconhecidos.

```
GNU nano 8.4 /etc/apache2/mods-enabled/security2.conf *
<IfModule security2_module>
    # Default Debian dir for modsecurity's persistent data
    SecDataDir /var/cache/modsecurity

    # Include all the *.conf files in /etc/modsecurity.
    # Keeping your local configuration in that directory
    # will allow for an easy upgrade of THIS file and
    # make your life easier
    #IncludeOptional /etc/modsecurity/*.conf

    # Include OWASP ModSecurity CRS rules if installed
    #IncludeOptional /usr/share/modsecurity-crs/*.load

    Include /usr/share/modsecurity-crs/rules/*.conf
    Include /etc/modsecurity/crs/crs-setup.conf
    Include /etc/modsecurity/crs/REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf
    Include /etc/modsecurity/crs/RESPONSE-999-EXCLUSION-RULES-AFTER-CRS.conf

</IfModule>
```

Assim, foi adicionado uma configuração *VirtualHost* na pasta de sites-available, owaspjs.conf. Esta redireciona o tráfego HTTP do domínio owaspjs.local para a aplicação Juice Shop (<http://10.0.0.2:3000>), utilizando os módulos proxy e proxy\_http.

```
GNU nano 8.4 /etc/apache2/sites-available/owaspjs.conf
<VirtualHost *:80>
    ServerName juiceshop.local

    ProxyPreserveHost On
    ProxyPass / http://10.0.0.2:3000/
    ProxyPassReverse / http://10.0.0.2:3000/

    SecRuleEngine On
</VirtualHost *:80>
```

Na figura abaixo, é possível observar a execução dos comandos utilizados para:

- Ativar os módulos necessários (proxy, proxy\_http e security2);
- Ativar o ficheiro de configuração owaspjs.conf com o comando a2ensite;
- Desativar o site predefinido (000-default.conf) para evitar conflitos de configuração;
- Reiniciar e recarregar o Apache2 para aplicar todas as alterações efetuadas.

```
(admin_sti@admin)-[~]
$ sudo a2enmod proxy
[sudo] senha para admin_sti:
Module proxy already enabled

(admin_sti@admin)-[~]
$ sudo a2enmod proxy_http
Considering dependency proxy for proxy_http:
Module proxy already enabled
Module proxy_http already enabled

(admin_sti@admin)-[~]
$ sudo a2enmod security2
Considering dependency unique_id for security2:
Module unique_id already enabled
Module security2 already enabled

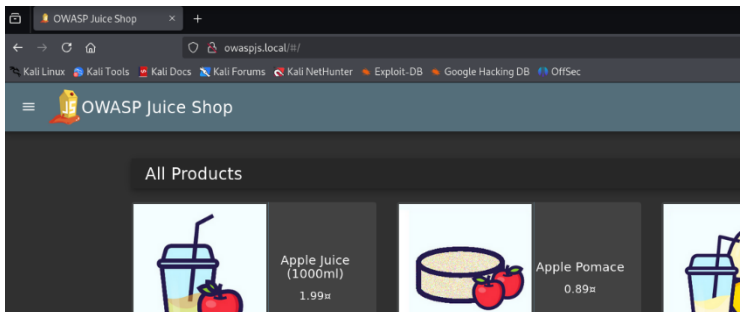
(admin_sti@admin)-[~]
$ sudo a2ensite owaspjs.conf
Site owaspjs already enabled

(admin_sti@admin)-[~]
$ sudo systemctl restart apache2

(admin_sti@admin)-[~]
$ sudo a2disable 000-default.conf
sudo: a2disable: comando não encontrado

(admin_sti@admin)-[~]
$ sudo a2dissite 000-default.conf
Site 000-default already disabled

(admin_sti@admin)-[~]
$ sudo systemctl reload apache2
```



```
(admin_sti@admin)-[~]
$ sudo tail -f /var/log/apache2/waf_error.log
[Wed May 28 17:33:40.715474 2025] [security2:error] [pid 3337:tid 3337] [client 10.0.0.3:33604] ModSec
nf"] [line "92"] [id "980130"] [msg "Inbound Anomaly Score Exceeded (Total Inbound Score: 5 - SQLI=0,X
lation") [hostname "owaspjs.local"] [uri "/socket.io/"] [unique_id "aDc65J4_oI1jAmyJ8YJGbwAAAAI"], ref
[Wed May 28 17:33:40.745487 2025] [security2:error] [pid 3337:tid 3337] [client 10.0.0.3:33604] ModSec
-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "956"] [id "920420"] [msg "Request content ty
uage-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "paranoia-level/1"] [tag "OWASP_CRS"]
referrer: http://owaspjs.local/
[Wed May 28 17:33:40.749202 2025] [security2:error] [pid 3337:tid 3337] [client 10.0.0.3:33604] ModSec
ST-949-BLOCKING-EVALUATION.conf"] [line "94"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Tota
ag "attack-generic") [hostname "owaspjs.local"] [uri "/socket.io/"] [unique_id "aDc65J4_oI1jAmyJ8YJGbw
[Wed May 28 17:33:40.749744 2025] [security2:error] [pid 3337:tid 3337] [client 10.0.0.3:33604] ModSec
nf"] [line "92"] [id "980130"] [msg "Inbound Anomaly Score Exceeded (Total Inbound Score: 5 - SQLI=0,X
lation") [hostname "owaspjs.local"] [uri "/socket.io/"] [unique_id "aDc65J4_oI1jAmyJ8YJGbwAAAAI"], ref
[Wed May 28 17:33:42.220829 2025] [security2:error] [pid 3947:tid 3947] [client 10.0.0.3:58720] ModSec
-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "956"] [id "920420"] [msg "Request content ty
uage-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "paranoia-level/1"] [tag "OWASP_CRS"]
referrer: http://owaspjs.local/
[Wed May 28 17:33:42.224556 2025] [security2:error] [pid 3947:tid 3947] [client 10.0.0.3:58720] ModSec
ST-949-BLOCKING-EVALUATION.conf"] [line "94"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Tota
ag "attack-generic") [hostname "owaspjs.local"] [uri "/socket.io/"] [unique_id "aDc65sUqG70L-0wEPLZyU"]
```

### 3. Web Application Security Testing

Todos os resultados obtidos serão anexados em anexos no fim do relatório com *printscreens* que achamos que são mais relevantes e os ficheiros csv serão enviados em conjunto com o relatório numa pasta “Resultados Testes (csv)”.

#### 3.0. Phase 1

##### Automated Scan

O scan inicial foi executado segundo o modo de ataque automático do ZAP, selecionando a funcionalidade “Automated Scan”. Este modo combina duas fases principais: o *spidering* e o *passive scanning*.

O *spidering* consiste em percorrer a aplicação web através do URL inicial fornecido de forma a descobrir todas as páginas, formulários, parâmetros e recursos acessíveis, construindo assim um mapa da aplicação.

Por outro lado, o *passive scanning* é a análise passiva realizada pelo ZAP sobre as requisições e respostas capturadas. Esta análise não interfere com a aplicação e procura identificar más práticas ou configurações inseguras, como cabeçalhos HTTP ausentes, exposição de informações sensíveis e utilização insegura de cookies.

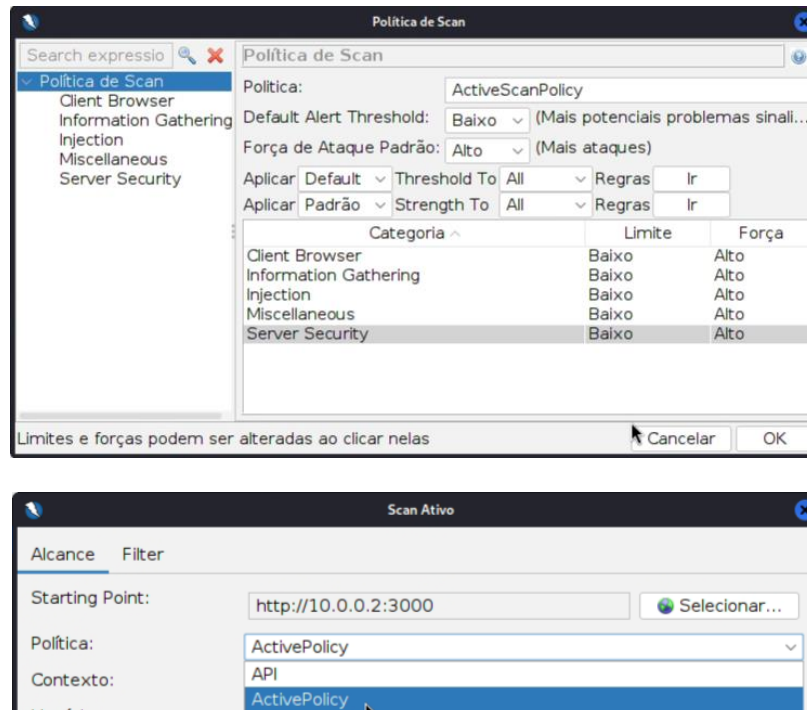


#### Resultados em Anexo A1 Resultados do Automated Scan

Como é possível observar pelos resultados obtidos, tanto o *spidering* e o *passive scanning* foram bem-sucedidos, conseguindo, assim, recolher URLs e alertas de segurança pertencentes à aplicação web.

## Active Scan

Este primeiro scan ativo é executado no URL inicial (http://10.0.0.2:3000) sem qualquer autenticação efetuada. Antes de ser executado foi criada uma política nova que intensifica a força dos ataques e baixa a tolerância a alertas encontrados.



## Resultados em Anexo A2 Resultados do Active Scan

Através dos dados recolhidos, foi possível observar uma diferença significativa entre o *automated scan* e o *active scan*. Ao contrário do *automated scan*, que inclui automaticamente uma fase de *spidering* para mapear a aplicação antes de iniciar os testes de segurança, o *active scan* personalizado foi executado sem qualquer *spidering* adicional, atuando unicamente sobre os pontos de entrada previamente identificados. Esta abordagem permitiu uma análise mais focada, baseada apenas nas áreas consideradas críticas ou mais propensas a vulnerabilidades, conforme definido na política de scan criada manualmente.

## Ataque Fuzzer

O *fuzzer* é uma ferramenta que o ZAP oferece através do *marketplace* instalado como uma extensão (*add-on*). Este consiste numa técnica de testes automáticos de segurança que envia dados aleatórios ou malformados para a aplicação com objetivo de descobrir vulnerabilidades ou falhas de funcionamento.



Neste caso, o *fuzzer* foi usado para atacar a página de *login* com o intuito de obter informações sobre a mesma. Para descobrir os POSTs que descrevem o login foi feito primeiramente um *manual explore* onde inserimos dados aleatórios nos campos e-mail e palavra-passe para estes apareçam no Zap.

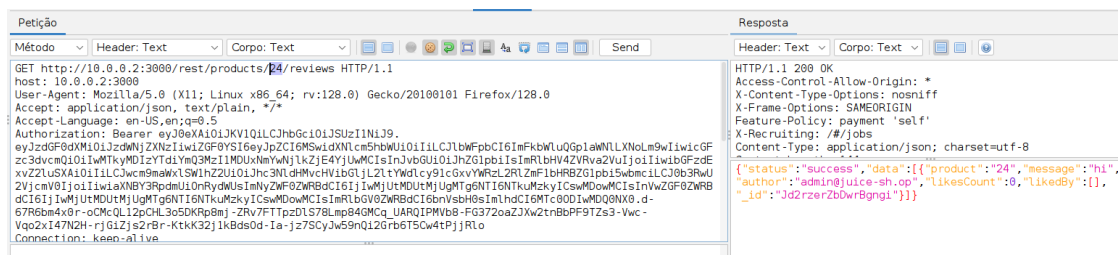


Com os POSTs recolhidos, fizemos então o fuzzer (configurado com SQLi e XSS) nos mesmos e conseguimos efetuar o login várias vezes. Verificamos que principalmente o uso de SQLinjection no parâmetro de email era o mais eficaz e onde o sistema apresentava uma maior vulnerabilidade.

Resultados em **Anexo A3** Resultados do Fuzzer

## Manual Penetration

Durante a realização da manual penetration, foi utilizada a requisição HTTP do tipo GET para aceder ao endpoint responsável pela visualização das reviews. No decorrer da análise, modificámos manualmente o parâmetro ID presente na URL da requisição. Observámos que a aplicação respondeu com os dados correspondentes ao novo ID fornecido.



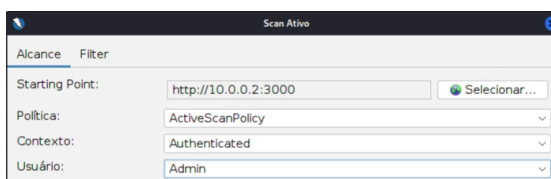
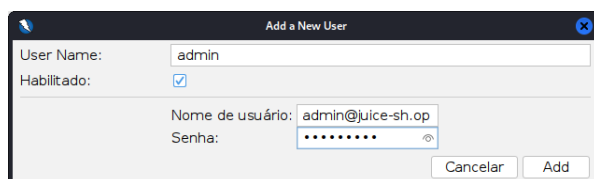
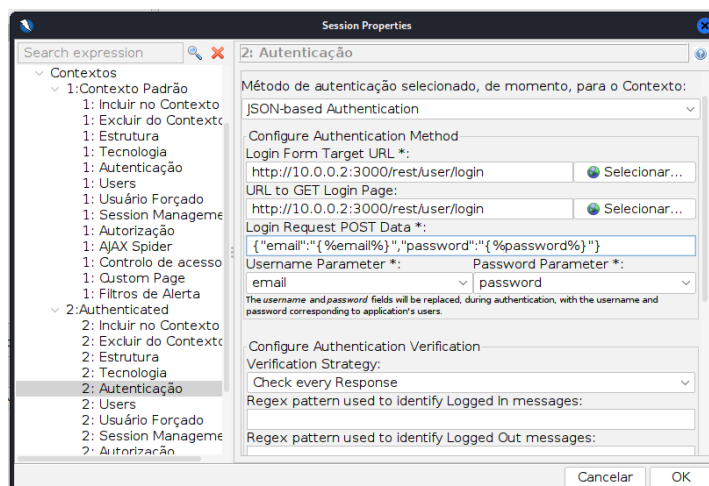
Este comportamento indica uma falta de validação de permissões no acesso aos recursos, permitindo a qualquer utilizador visualizar diferentes aspetos do website apenas alterando o ID na requisição.

## Active Scan (Authenticated)

Com os dados obtidos dos testes anteriores, conseguimos entrar dentro da aplicação como administrador, obtendo acesso a páginas e dados inalcançáveis anteriormente.

Assim, com este acesso ao modo administrativo, aplicamos outro teste *Active Scan* com objetivo de identificar vulnerabilidades que apenas se manifestam quando o utilizador possui privilégios elevados. Este tipo de análise é essencial para avaliar a robustez da aplicação em cenários de utilização real, onde utilizadores autenticados podem representar uma ameaça interna ou ser alvo de ataques que exploram permissões excessivas ou indevidamente atribuídas.

A configuração para este teste foi aplicada do seguinte modo:



Resultados em **Anexo A4** Resultados do Active Scan (Authenticated)

## 3.1. Information Gathering

- 1) Conduct Search Engine Discovery Reconnaissance for Information Leakage

De acordo com o “OWASP Web Security Testing Guide”, uma das etapas iniciais na avaliação da segurança de aplicações web consiste na realização de reconhecimento através de motores de busca. Esta técnica visa identificar informações sensíveis que possam estar expostas na internet, tanto em sites próprios da organização como em plataformas de terceiros.

Este processo de conhecimento pode ser dividido em dois métodos: métodos diretos – envolvem a pesquisa nos índices dos motores de busca e nos conteúdos armazenados em cache; métodos indiretos – consistem na procura de informações sensíveis em fóruns, grupos de discussão e sites de terceiros.

Em relação aos métodos diretos, já obtemos dados destes com o uso do *spidering* e *passive/active scan* onde encontramos páginas admin, *pdfs* associados e diretórios *index*. Ao contrário deste, os métodos indiretos não foram possíveis pois como este é um servidor local não conseguimos efetuar pesquisa em sites de terceiros que redirecionassem a este.

- 2) Fingerprint Web Server

Este teste foi efetuado, como sugerido pelo guia do OWASP, usando comandos como *curl* diretamente na máquina do atacante com objetivo de recolher cabeçalhos HTTP expostos pelo servidor. Os resultados obtidos encontram-se apresentados abaixo:

```
(admin_sti@admin)-[~]
$ curl -I http://10.0.0.2:3000/
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, 25 May 2025 18:53:10 GMT
ETag: W/"138f5-19708caaf6d"
Content-Type: text/html; charset=UTF-8
Content-Length: 80117
Vary: Accept-Encoding
Date: Sun, 25 May 2025 19:05:58 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

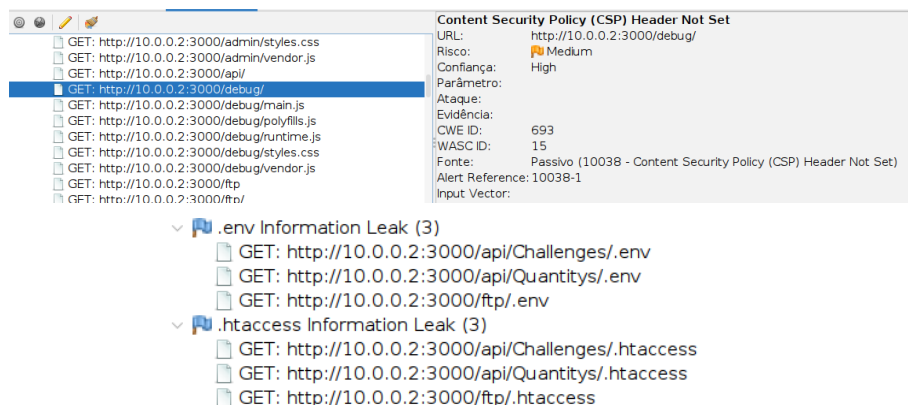
## 3.2. Configuration and Deployment Management Testing

### • 2) Test Application Platform Configuration

Este teste tem como objetivo analisar a configuração da plataforma onde a aplicação web se encontra alojada, de forma a identificar possíveis falhas de segurança resultantes de definições incorretas ou omissões na configuração. Conforme proposto no guia OWASP Web Security Testing Guide, é essencial verificar se a plataforma (por exemplo, o servidor de aplicação, *frameworks* ou bibliotecas de suporte) está devidamente atualizada, com funcionalidades desnecessárias desativadas e sem informações sensíveis expostas. Configurações incorretas podem revelar dados técnicos, ficheiros de *debug*, diretórios ocultos, interfaces administrativas ou versões vulneráveis de componentes utilizados pela aplicação.

Para validar a existência destas falhas de segurança foram testados entes *endpoints* comuns numa configuração de uma aplicação web:

- `http://10.0.0.2:3000/admin/`
- `http://10.0.0.2:3000/test/`
- `http://10.0.0.2:3000/debug/`
- `http://10.0.0.2:3000/api/`
- `http://10.0.0.2:3000/.git/`
- `http://10.0.0.2:3000/.env/`



Como é possível observar pelas figuras acima, existem ficheiros em *endpoints* HTTP públicos que representam uma falha de configuração crítica, como por exemplo: páginas de *debug*, ficheiros que contêm informações sensíveis podendo levar ao acesso indesejado de informações sobre a infraestrutura da aplicação, caminhos internos do sistema, entre outros.

- **3) Test File Extensions Handling for Sensitive Information**

Através da utilização do comando cURL, foi possível verificar que ficheiros como o connection.inc, ao serem acedidos diretamente, são redirecionados automaticamente para a página inicial em HTML da aplicação Juice Shop.

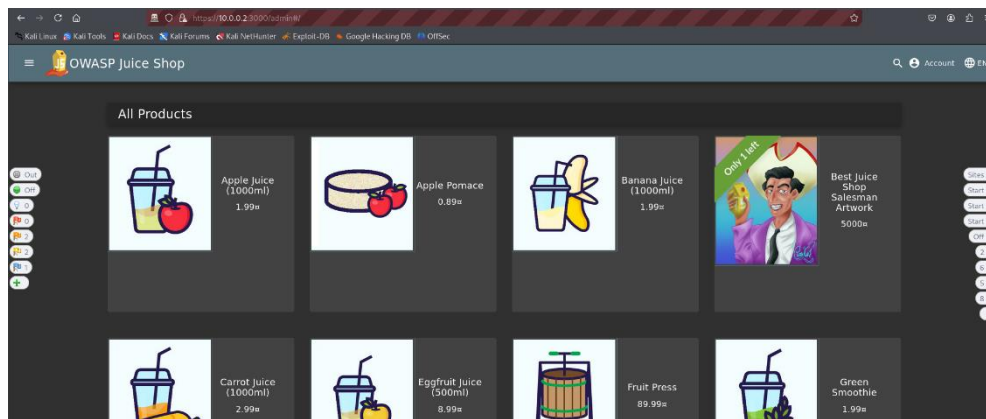
```
(admin_sti@admin)-[~]
$ curl -i http://10.0.0.2:3000/connection.inc

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: Mon, 26 May 2025 13:12:21 GMT
ETag: W/"138f5-1970cb9041d"
Content-Type: text/html; charset=UTF-8
Content-Length: 80117
Vary: Accept-Encoding
Date: Mon, 26 May 2025 13:45:42 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

Este comportamento indica que, embora os ficheiros possam ser referenciados, o servidor está configurado para redirecionar acessos não autorizados ou diretos, evitando a sua exposição direta a qualquer utilizador.

- **5) Enumerate Infrastructure and Application Admin Interfaces**

Este teste tem como objetivo identificar interfaces administrativas expostas que possam permitir o acesso não autorizado total à configuração da aplicação ou da sua infraestrutura. Interfaces deste tipo, como /admin ou /dashboard, se acessíveis, podem representar um risco crítico.



Durante o processo, foram testados diversos endpoints comuns associados a páginas administrativas, como *admin*, *administrator*, *dashboard*, entre outros. Todos os pedidos realizados resultaram em redirecionamentos para a página inicial da aplicação, o que indica que essas páginas não estão disponíveis ou estão devidamente protegidas contra acessos diretos não autenticados.

## • 6) Test HTTP Methods

Neste teste testamos os métodos disponibilizados pelo HTTP, nomeadamente GET, POST, PUT, DELETE.

```
GET http://10.0.0.2:3000/ HTTP/1.1
host: 10.0.0.2:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
```

```
PUT http://10.0.0.2:3000/ HTTP/1.1
host: 10.0.0.2:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
content-length: 0
```

```
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

<!--
  ~ Copyright (c) 2014-2025 Bjoern
  ~ SPDX-License-Identifier: MIT
-->
```

```
<!doctype html>
<html lang="en" data-beasties-cont
<head>
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN

<!--
  ~ Copyright (c) 2014-2025 Bjoern Kimminich
  ~ SPDX-License-Identifier: MIT
-->

<!doctype html>
<html lang="en" data-beasties-container>
<head>
<meta charset="utf-8">
<title>OWASP Juice Shop</title>
<meta name="description" content="Probab
<meta name="viewport" content="width=device
```

```
DELETE http://10.0.0.2:3000/ HTTP/1.1
host: 10.0.0.2:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
content-length: 0
```

```
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-Range: bytes

<!--
  ~ Copyright (c) 2014-2025 Björn Kimminich
  ~ SPDX-License-Identifier: MIT
-->

<!doctype html>
<html lang="en" data-beasties-container>
<head>
  <meta charset="utf-8">
  <title>OWASP Juice Shop</title>
```

```
POST http://10.0.0.2:3000/ HTTP/1.1
host: 10.0.0.2:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
content-length: 0
```

```
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-Range: bytes

<!--
  ~ Copyright (c) 2014-2025 Björn Kimminich
  ~ SPDX-License-Identifier: MIT
-->

<!doctype html>
<html lang="en" data-beasties-container>
<head>
  <meta charset="utf-8">
  <title>OWASP Juice Shop</title>
  <meta name="description" content="Pr
```

Como demonstrado nas figuras acima, a aplicação aceitou a todos os métodos testados. A aceitação de métodos como PUT e DELETE, sem controlo de acesso adequado, podendo permitir a manipulação ou remoção indevida de recursos, representando uma vulnerabilidade crítica.

## • 7) Test HTTP Strict Transport Security

Este teste tem como finalidade verificar se a aplicação implementa a política de segurança HTTP Strict Transport Security (HSTS). Esta política obriga os *browsers* a comunicarem exclusivamente por HTTPS, prevenindo diverse tipos de ataques.

Para realizar o teste, foi utilizado o seguinte comando:

```
(admin_sti@admin)-[~]
$ curl -s -D- https://10.0.0.2:3000 | grep -i strict
```

Como resultado não foi obtido nada em resposta ao mesmo, o que indica a inexistência deste protocolo.

## • 8) Test RIA Cross Domain Policy

Para este teste o comportamento é igual ao que se sucedeu no teste 3.2.3, ambos ficheiros xml são direcionados para a pagina html inicial da aplicação.



```

$ curl -s http://10.0.0.2:3000/crossdomain.xml
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=8
Last-Modified: Mon, 26 May 2025 13:12:21 GMT
ETag: W/"138f5-1970cb9041d"
Content-Type: text/html; charset=UTF-8
Content-Length: 8817
Vary: Accept-Encoding
Date: Mon, 26 May 2025 15:52:36 GMT
Connection: keep-alive
Keep-Alive: timeout=5

<!--
~ Copyright (c) 2014-2025 Björn Kimminich & the OWASP Juice Shop contributors.
~ SPDX-License-Identifier: MIT

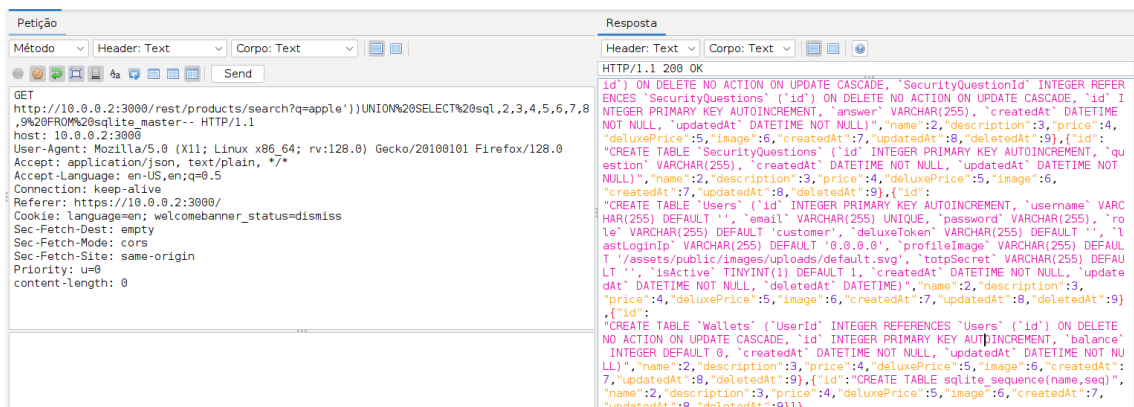
-->
<!-->
<doctype html>
<html lang="en" data-beasties-container>
<head>
  <meta charset="utf-8">
  <title>OWASP Juice Shop</title>
  <meta name="description" content="Probably the most modern and sophisticated insecure web application">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="favicon" href="/assets/favicon.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>
</head>
<body>
  window.addEventListener('load', function() {
    window.cookieconsent.initialise({
      "palette": {
        "popup": { "background": "var(--theme-primary)", "text": "var(--theme-text)" },
        "button": { "background": "var(--theme-accent)", "text": "var(--theme-text)" }
      },
      "theme": "classic",
      "position": "bottom-right",
      "content": "This website uses fruit cookies to ensure you get the juiciest tracking experience.", "dim
  });
</body>

```



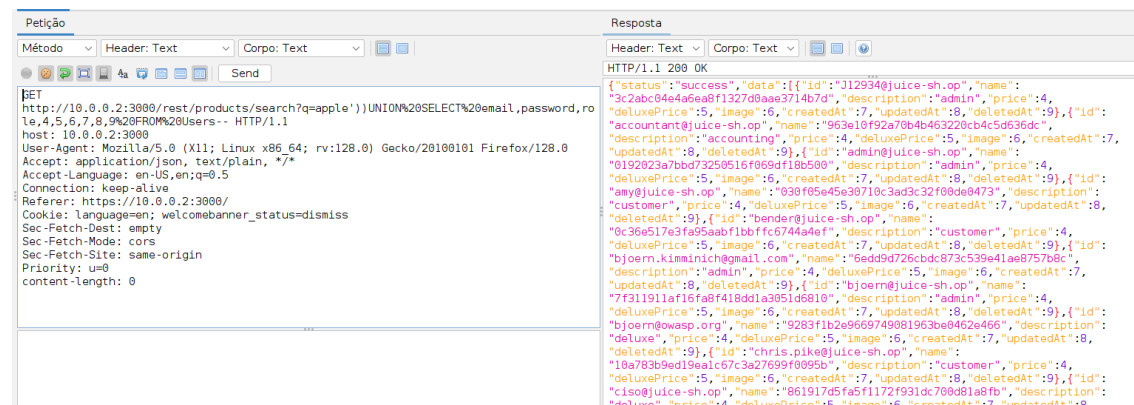
Com base neste ponto de entrada, foram injetados erros propositalmente em pedidos HTTP, enviados pelo *requester* do ZAP, com o objetivo de identificar o sistema de gestão de base de dados utilizado. A resposta da aplicação revelou que o sistema em uso era o SQLite, uma vez que a estrutura SQLite Master foi devolvida em mensagens de erro.

Sabendo, então, que programa era usado manipulamos a pesquisa de forma a enviar pedidos SQL como SELECT e UNION SELECT para descobrir que tabelas existentes na aplicação, como é possível observar na figura abaixo:



Como ilustrado na figura acima, foi possível descobrir a existência de uma tabela denominada “Users”, contendo colunas como email, password e role.

Com base nestas informações, foi efetuada uma nova injeção SQL com o objetivo de obter os dados de todos os utilizadores registados na aplicação, incluindo os respetivos emails, palavras-passe e papéis atribuídos. Este resultado demonstrou a presença de utilizadores com diferentes níveis de privilégio, validando a implementação de papéis distintos na aplicação, como “Customer”, “Admin”, “Deluxe” e “Accountant”.

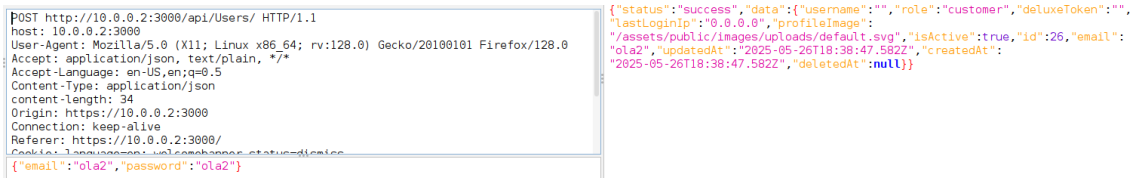


Para uma melhor visualização dos JSON recolhidos com o SQLi, veja [users.json](#) e [sqi-tables.json](#).

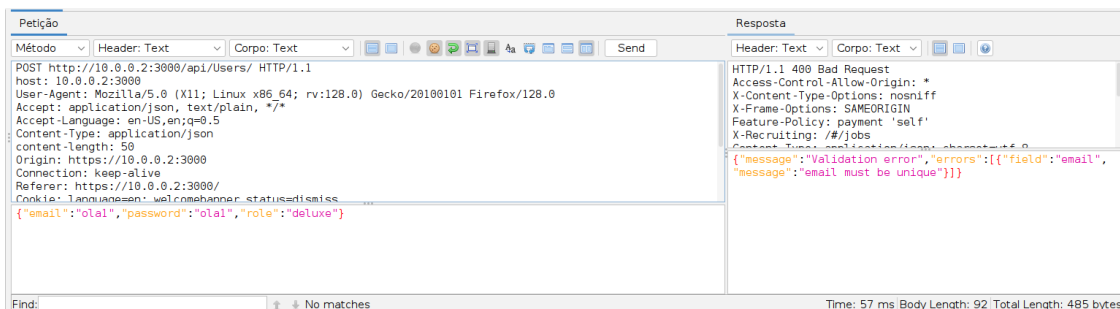
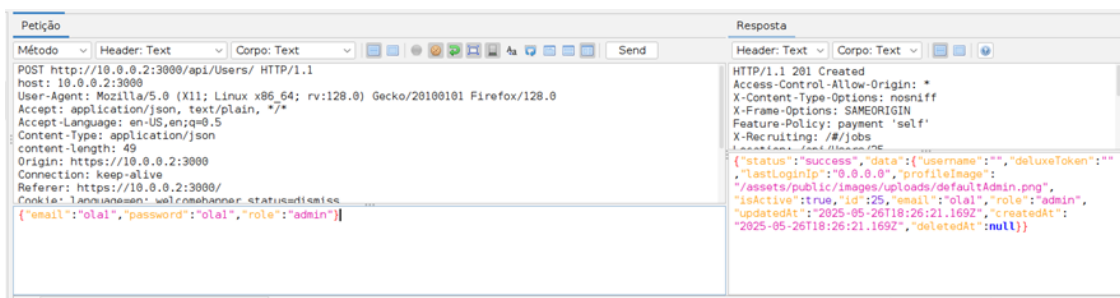
## • 2) Test User Registration Process

Este teste tem como objetivo analisar a segurança durante o processo de criação de novos utilizadores numa aplicação web.

Durante o teste, procuramos identificar se existem falhas que permitam, por exemplo, a reutilização de endereços de email já registados, a criação de utilizadores com permissões elevadas sem controlo adequado, ou a omissão de mecanismos de verificação, como a confirmação por email.



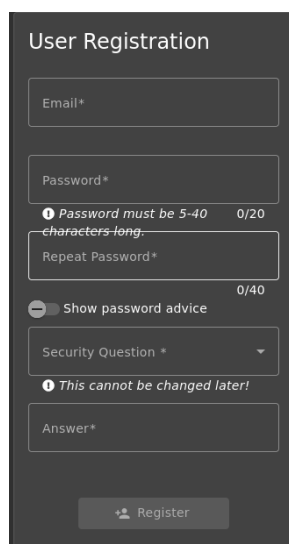
Na figura acima podemos ver a criação de um utilizador “ola2” com uma password extremamente fraca através do ZAP e a resposta de sucesso da aplicação. Assim, verificamos tanto a falta de utilização de *standards* de segurança, como palavras-passe iguais ou maiores que 8 dígitos, uso de caracteres especiais e/ou confirmação via e-mail.



Em relação às figuras previamente apresentadas, podemos ver a criação de um utilizador, novamente com introdução de campos com falhas de segurança, com uma role definida. Como se pode verificar, a aplicação aceitou o pedido sem validação adequada, permitindo a criação de um utilizador com um perfil potencialmente privilegiado — o que representa uma falha grave de segurança.

```
"deluxePrice":5,"image":6,"createdAt":7,"updatedAt":8,"deletedAt":9},{ "id":  
"morty@juice-sh.op","name":"f2f933d0bb0ba057bc8e33b8ebd6d9e8","description":  
"customer","price":4,"deluxePrice":5,"image":6,"createdAt":7,"updatedAt":8,  
"deletedAt":9},{ "id":"ola","name":"2fe04e524ba40505a82e03a2819429cc",  
"description":"customer","price":4,"deluxePrice":5,"image":6,"createdAt":7,  
"updatedAt":8,"deletedAt":9},{ "id":"ola1","name":  
"e39c472d686c4aa328e100fd934b3657","description":"admin","price":4,  
"deluxePrice":5,"image":6,"createdAt":7,"updatedAt":8,"deletedAt":9},{ "id":  
"ola2","name":"68fc9de90e2658c80a41fb41a59ee6f2","description":"customer",  
"price":4,"deluxePrice":5,"image":6,"createdAt":7,"updatedAt":8,"deletedAt":9}  
, {"id":"ola@example","name":"4230f911529de78e8e012e7b6745a40c","description":  
"customer","price":4,"deluxePrice":5,"image":6,"createdAt":7,"updatedAt":8,  
"deletedAt":9},{ "id":"stan@juice-sh.op","name":  
"e9048a3f43dd5e094ef733f3bd88ea64","description":"deluxe","price":4,  
"deluxePrice":5,"image":6,"createdAt":7,"updatedAt":8,"deletedAt":9},{ "id":  
"support@juice-sh.op","name":"3869433d74e3d0c86fd25562f836bc82","description":
```

Para além das falhas demonstradas acima, verificamos que durante o processo de criação de um novo utilizador não é pedido um nome de utilizador, mas como podemos ver nos testes realizados para as “Test Role Definitions” cada conta contém um *username*. Assim, concluímos que os nomes do utilizador não são criados pelo utilizador, mas sim pela aplicação.



The image shows a 'User Registration' form with the following fields and features:

- Email\***: A text input field.
- Password\***: A text input field with a feedback message: 'Password must be 5-40 characters long.' and a character count '0/20'.
- Repeat Password\***: A text input field with a character count '0/40'.
- Show password advice**: A toggle switch.
- Security Question \***: A dropdown menu with a warning message: 'This cannot be changed later!'.
- Answer\***: A text input field.
- Register**: A button with a user icon.

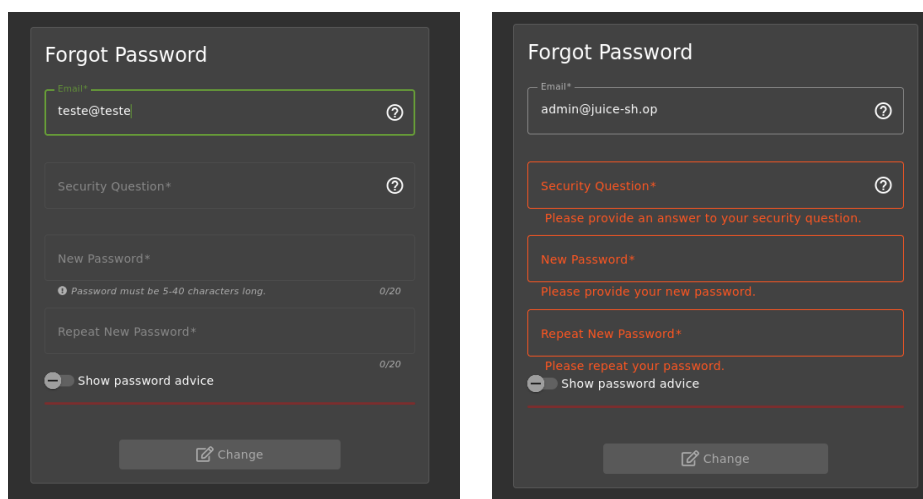
### • 3) Test Account Provisioning Process

Este teste reflete os mesmos resultados que o anterior.

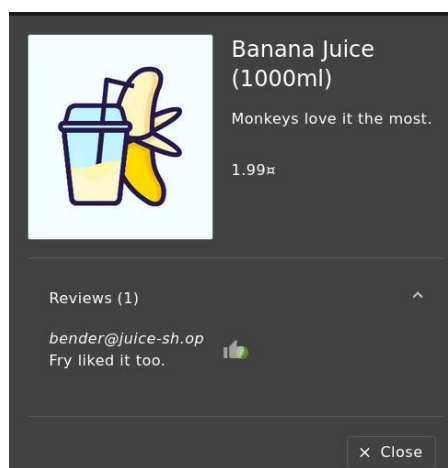
- 4) Testing for Account Enumeration and Guessable User Account

Este teste visa identificar se a aplicação permite a enumeração de contas de utilizador através de mensagens de erro distintas ou da análise de respostas dadas pela aplicação durante os processos de autenticação e recuperação de conta.

Observou-se, através da funcionalidade "Esqueci-me da palavra-passe", que a aplicação revela implicitamente a existência de contas de utilizador. Quando é introduzido um endereço de e-mail válido, o campo correspondente é realçado a verde, indicando que o e-mail está registado no sistema. Por outro lado, ao introduzir um endereço inválido, os campos ficam a vermelho, sinalizando que não correspondem a uma conta existente. Este comportamento permite a enumeração de contas por parte de um atacante, constituindo uma vulnerabilidade de segurança relevante.



Conseguimos, também, verificar que a aplicação expõe os e-mails dos utilizadores ao longo da sua interface invés de *usernames*. Esta prática representa um risco de segurança, uma vez que os endereços de e-mail são frequentemente utilizados como credenciais de login, tanto nesta como noutras aplicações. Assim, um atacante pode explorar esta exposição para realizar ataques de força bruta, comprometendo contas não só nesta aplicação, mas também em outros serviços onde o utilizador reutilize as mesmas credenciais.



- **5) Testing for Weak or Unenforced Username Policy**

Este teste tem como objetivo verificar se a aplicação impõe uma política robusta e coerente para a criação de nomes de utilizador. A ausência de restrições claras ou a aceitação de nomes de utilizador previsíveis, curtos ou malformados pode comprometer a segurança geral do sistema de autenticação.

Como foi verificado nos testes anteriores, não existe nenhum processo de criação de *username*, eles são atribuídos pela aplicação e não são apresentados em qualquer ponto do *website*, apenas os e-mails.

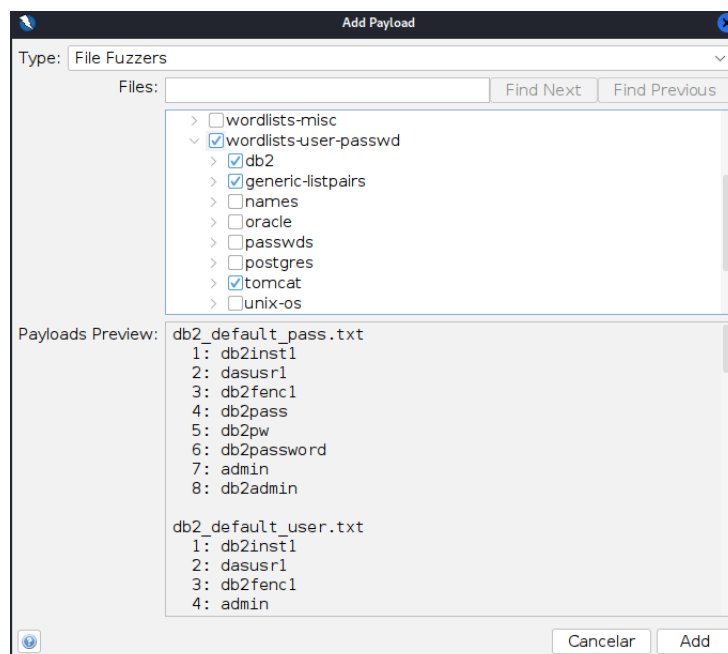
### 3.4. Authentication Testing

- **1) Testing for Credentials Transported over an Encrypted Channel**

Foi verificado no teste 3.2.7 “Test HTTP Strict Transport Security” que a aplicação não usa o protocolo HTTPS.

- **2) Testing for Default Credentials**

Este teste tem como objetivo verificar se a aplicação estão a utilizar credenciais por omissão (*default credentials*), frequentemente deixadas ativas após a instalação inicial de sistemas, *frameworks* ou serviços.

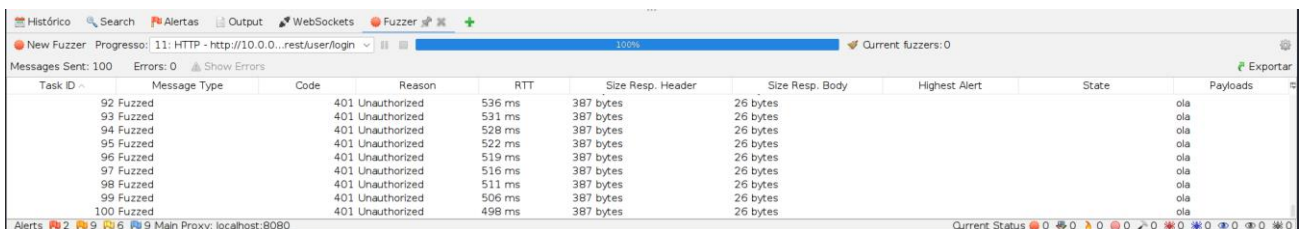


Usamos os add-ons previamente instalados para fazer uso de emails e passwords default e verificamos que o sistema não era vulnerável às mesmas.

### • 3) Testing for Weak Lock Out Mechanism

Este teste tem como finalidade avaliar se a aplicação implementa mecanismos de bloqueio após múltiplas tentativas de autenticação falhadas. A ausência de um sistema de bloqueio permite que atacantes realizem ataques de força bruta, tentando adivinhar palavras-passe sem restrições, o que compromete gravemente a segurança das contas de utilizador.

Durante a realização deste teste, simulamos um número elevado de tentativas de login falhadas consecutivas com a utilização do *fuzzer*, para avaliar se a aplicação reage de forma adequada. Pelos resultados obtidos, concluímos que esta não contém nenhum tipo de mecanismo de bloqueio para este caso.



Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
92	Fuzzed	401	Unauthorized	536 ms	387 bytes	26 bytes			ola
93	Fuzzed	401	Unauthorized	531 ms	387 bytes	26 bytes			ola
94	Fuzzed	401	Unauthorized	528 ms	387 bytes	26 bytes			ola
95	Fuzzed	401	Unauthorized	522 ms	387 bytes	26 bytes			ola
96	Fuzzed	401	Unauthorized	519 ms	387 bytes	26 bytes			ola
97	Fuzzed	401	Unauthorized	516 ms	387 bytes	26 bytes			ola
98	Fuzzed	401	Unauthorized	511 ms	387 bytes	26 bytes			ola
99	Fuzzed	401	Unauthorized	506 ms	387 bytes	26 bytes			ola
100	Fuzzed	401	Unauthorized	498 ms	387 bytes	26 bytes			ola

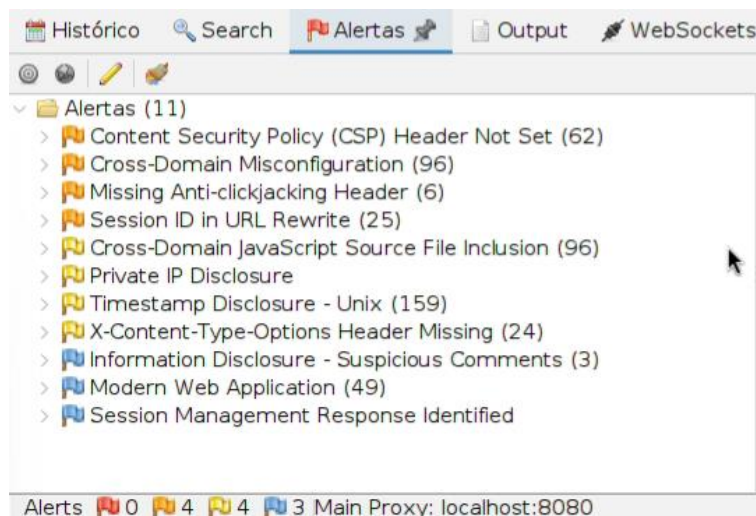
## 4. Conclusão

Através da realização deste trabalho foi possível compreender, na prática, o funcionamento de uma auditoria de segurança a uma aplicação web, aplicando os princípios do OWASP Web Security Testing Guide. A utilização da ferramenta OWASP ZAP revelou-se fundamental para a identificação de múltiplas vulnerabilidades, desde falhas básicas como a aceitação de métodos HTTP indevidos, até questões mais críticas como a exposição de contas administrativas, entre outros.

No geral, este trabalho contribuiu significativamente para o desenvolvimento de competências práticas em segurança ofensiva web, bem como para a consolidação de conhecimentos teóricos sobre vulnerabilidades e medidas de mitigação em aplicações web modernas.

# Anexos

## Anexo A1 Resultados do Automated Scan

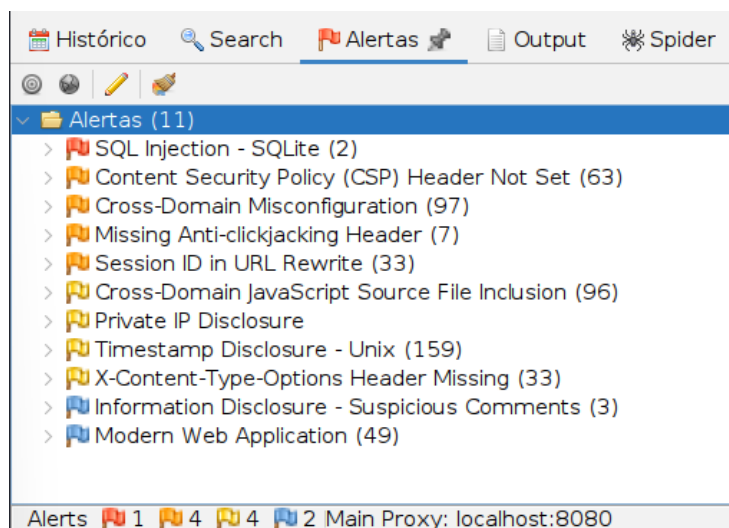


**Ficheiros CSV:**

[Spidering.csv](#)

[PassiveScan.csv](#)

## Anexo A2 Resultados do Active Scan



**Ficheiros CSV:**

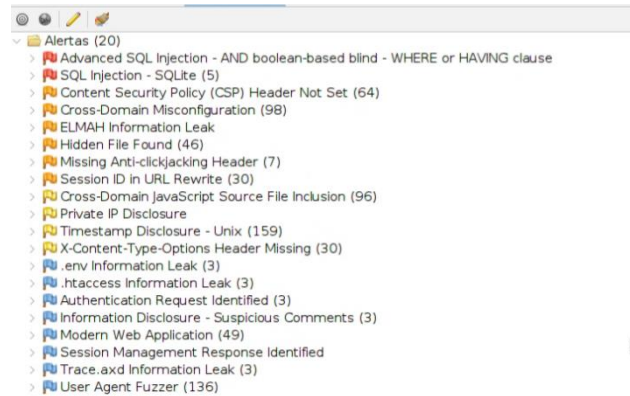
[Non-auth-ActiveScan.csv](#)

## Anexo A3 Resultados do Fuzzer

### Ficheiros CSV:

[Fuzzer.csv](#)

## Anexo A4 Resultados do Active Scan (Authenticated)



### Ficheiros CSV:

[ActiveScanAuthenticated.csv](#)