

# Practical Assignment #3

João Neto – 2023234004

Vasco Alves – 2022228207

31 de maio de 2026

# Índice

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Architecture Considered for Both Stages</b>	<b>3</b>
2.1	Network structure . . . . .	3
2.2	Servers . . . . .	4
2.3	Services . . . . .	4
<b>3</b>	<b>Web application security testing</b>	<b>4</b>
3.1	Information Gathering . . . . .	4
3.2	Configuration and Deployment Management Testing . . . . .	5
3.3	Identity Management Testing . . . . .	5
3.4	Authentication Testing . . . . .	7
3.5	Authorization Testing . . . . .	7
3.6	Session Management Testing . . . . .	7
3.7	Input Validation Testing . . . . .	8
3.7.1	Testing for SQL Injection . . . . .	8
3.8	Testing for Error Handling . . . . .	9
3.9	Client Side Testing . . . . .	9
<b>4</b>	<b>Web Application Security Firewall</b>	<b>10</b>
4.1	Information Gathering . . . . .	10
4.2	Configuration and Deployment Management Testing . . . . .	10
4.3	Identity Management Testing . . . . .	10
4.4	Authentication Testing . . . . .	10
4.5	Authorization Testing . . . . .	10
4.6	Session Management Testing . . . . .	10
4.7	Input Validation Testing . . . . .	10
4.8	Testing for Error Handling . . . . .	10
4.9	Client Side Testing . . . . .	10
<b>5</b>	<b>Conclusions</b>	<b>10</b>

## 1 Introduction

Este trabalho tem como objetivo realizar testes de penetração numa aplicação cobaia (o *Juicebox*) desenhada para aprendizagem.

Este trabalho tem como objetivo utilizar o **WSTG** (Web security testing guide) e configurar um ModSecurity reverse proxy como uma **WAF**. Para esse fim temos uma aplicação cobaia (o *Juicebox*) desenhada para aprendizagem que vamos utilizar num ambiente controlado para aprender como descobrir vulnerabilidades (aplicando o **WSTG** e recorrendo ao **OWASP ZAP**) e prevenir antes do serviço estar online (elaborando uma **WAF**).

## 2 Architecture Considered for Both Stages

Utilizámos somente duas máquinas virtuais: um servidor a correr *CentOS 9* e um cliente a correr *Kali Linux*. O servidor contém o serviço *Apache*, que age como *firewall* através do módulo *ModSecurity*, e um servidor *Node.js* que aloja o *Juicebox* — a aplicação que vai servir de cobaia (*dummy*).

Com o ambiente criado foram realizadas duas etapas de testes:

- Primeira etapa: Explorar vulnerabilidades na aplicação que existem sem a **WAF**
- Segunda etapa: Verificar que vulnerabilidades foram mitigadas da primeira etapa com o uso de uma **WAF** configurada.

Realisticamente estas etapas podiam continuar a repetir-se, até que estivessemos satisfeitos com o resultado, mas para o fim deste projeto estas etapas serão suficientes.

### 2.1 Network structure

- **Client (20.60.0.0/24)** Cliente.
- **Server (10.60.0.0/24)** Apache+ModSecurity e JuiceShop.

### 2.2 Servers

- **10.60.0.1** Servidor CentOS 9 com WAF e aplicação JuiceShop.

### 2.3 Services

Service	Port
NodeJS (JuiceShop)	3000
Apache (WAF)	80

## 3 Web application security testing

### 3.1 Information Gathering

Utilizámos a política por omissão (*default policy*) para a realização do *Active Scan* através do OWASP ZAP. Com esta abordagem, obtivemos múltiplos alertas automáticos. De forma a priorizar a análise, investigamos as alertas principais com base no maior nível de risco e grau de confiança reportados pela ferramenta.

Para conseguir informação inicial realizamos um *Active Scan* através do OWASP ZAP, o policy utilizado para esse scan foi *Default Policy*. Foi obtido vários alertas automáticos devido a esse scan e decidimos investigar as alertas principais com base no nível de risco e grau de confiança reportado pela ferramenta.

Adicionalmente, realizámos testes de infraestrutura utilizando ferramentas especializadas:

```
bash
```

```
1 sqlmap -u "http://192.168.1.1:3000/rest/products/search?q=apple" -p q --  
   level=5 --risk=3 --banner
```

Ao executar o *sqlmap*, descobrimos que o sistema de gestão de base de dados subjacente é o *SQLite*.

Paralelamente, realizámos uma descoberta de ficheiros e diretórios através de técnicas de *fuzzing* de URLs no OWASP ZAP recorrendo à lista de permissões da *DirBuster*. Esta exploração revelou os seguintes endpoints publicamente expostos:

- `/ftp`: Servidor de armazenamento e transferência de ficheiros exposto.
- `/metrics`: Métricas internas da infraestrutura expostas.
- `/api-docs`: Documentação e esquemas estruturais da API.

### 3.2 Configuration and Deployment Management Testing

#### Enumerate Infrastructure and Application Admin Interfaces

Identificámos e testámos o acesso ao endpoint `/api-docs` (*Swagger UI*), validando que as interfaces de documentação interna do sistema e as definições da API estavam publicamente expostas sem qualquer tipo de controlo de acesso ou autenticação prévia.

#### Test HTTP Methods

Testámos os métodos HTTP permitidos pelo servidor através do envio de pedidos `OPTIONS`. Verificámos que o servidor aceita métodos potencialmente perigosos ou desnecessários para utilizadores comuns em rotas específicas, expandindo a superfície de ataque da aplicação.

## Test File Permission

Analisámos as permissões de acesso no diretório `/ftp`. Verificámos que a falta de restrições rígidas ao nível do sistema de ficheiros permite a qualquer utilizador anónimo listar o conteúdo de diretórios estruturais e descarregar ficheiros não indexados na interface principal da aplicação.

## 3.3 Identity Management Testing

### Test Role Definitions

Efetuámos testes de manipulação de parâmetros do lado do cliente através das ferramentas de programador do navegador. Adicionámos manualmente os cookies `isAdmin` com o valor `true` e `role` com o valor `admin`. Após a atualização da página, não observámos qualquer escalonamento de privilégios, indicando que a aplicação não valida perfis administrativos com base nestes cookies específicos.

### Test User Registration Process

Utilizámos o OWASP ZAP para intercetar o tráfego de rede e definir um *breakpoint* no pedido HTTP POST de registo de novos utilizadores. Modificámos o corpo do pedido JSON, injetando manualmente o parâmetro `"role":"admin"`:

```
json
1 {
2   "email": "johnGomas@gmail.com",
3   "role": "admin",
4   "password": "password",
5   "passwordRepeat": "password",
6   "securityQuestion": {
7     "id": 2,
8     "question": "Mother's maiden name?",
9     "createdAt": "2026-05-30T12:28:33.216Z",
10    "updatedAt": "2026-05-30T12:28:33.216Z"
11  },
12  "securityAnswer": "poker"
13 }
```

O servidor backend processou o pedido sem validar se o utilizador possuía autorização para definir o seu próprio perfil, o que resultou na criação bem-sucedida de uma conta com permissões totais de administrador (*Mass Assignment Vulnerability*).

### Testing for Account Enumeration and Guessable User Account

Ao tentar registar um utilizador com o e-mail `admin@juice-sh.op`, verificámos que a aplicação devolve uma mensagem de erro explícita indicando que o e-mail já se encontra registado no sistema. Este comportamento confirma a vulnerabilidade de enumeração de contas, permitindo a um atacante mapear quais os e-mails válidos na plataforma.

## User Registration

Email must be unique

Email\*

admin@juice-sh.op

Password\*

●●●●●●●●

*! Password must be 5-40 characters long.* 8/20

Repeat Password\*

●●●●●●●●

8/40

### Testing for Weak or Unenforced Username Policy

Após testar vários caracteres especiais no formulário de registo, criámos um utilizador com os seguintes dados nos campos de input:

- **E-mail:** son'or1=1-@gmail.com
- **Nome/Campos Adicionais:** <h1>STRONG

A aplicação aceitou o registo sem validar a presença de caracteres de injeção SQL ou tags HTML. Contudo, verificámos que é impossível efetuar login com esta conta posteriormente, uma vez que o processo de autenticação falha e resulta num erro genérico do tipo [object Object] no ecrã.

## Login

[object Object]

Email\*

son'or1=1--@gmail.com

Password\*

●●●●●●●●

[Forgot your password?](#)

☒ Remember me

[Not yet a customer?](#)

### 3.4 Authentication Testing

Realizámos testes de *fuzzing* automatizado contra o formulário de login utilizando dicionários de credenciais. Identificámos que a aplicação não implementa mecanismos de bloqueio de conta (\*Account Lockout\*) ou limitação de taxa de pedidos (\*Rate Limiting\*), permitindo ataques contínuos de *brute force*.

### 3.5 Authorization Testing

Testámos as permissões de acesso ao diretório `/ftp` e verificámos que o servidor está configurado para permitir nativamente apenas a visualização de ficheiros com as extensões `.md` e `.pdf`.

Seguidamente, explorámos falhas na validação de inputs através de uma injeção de *Null Byte* codificado (`%2500.md` ou `%2500.pdf`). O ataque foi bem-sucedido e contornou a validação de extensões do servidor, garantindo o acesso e descarregamento de ficheiros confidenciais restritos: `encrypt.pyc` e `suspicious_errors.yml`.

### 3.6 Session Management Testing

Identificámos que o cookie `token`, responsável por armazenar o identificador da sessão ativa do utilizador, possui a flag `HttpOnly` configurada como `false`. A ausência desta proteção significa que o token está totalmente exposto e pode ser lido por scripts do lado do cliente, tornando a sessão criticamente vulnerável a roubo por Cross-Site Scripting (XSS).

### 3.7 Input Validation Testing

#### Testing for Reflected Cross Site Scripting

Durante a auditoria à barra de pesquisa de produtos, validámos a existência de uma vulnerabilidade de *Reflected Cross-Site Scripting* (XSS) devido à ausência de higienização do input do utilizador.

1. **Injeção HTML:** Introduzimos o valor `<h1>apple` na pesquisa e verificámos que o resultado foi renderizado no navegador como um título estrutural, confirmando que o código HTML é injetado diretamente na página.
2. **Tentativa com Script Direto:** Inserimos o payload tradicional `<script>alert('someones gotta do it')</script>apple`. Esta tentativa não foi executada, demonstrando a presença de uma validação simples contra tags explícitas de script.
3. **Evasão com Evento de Erro:** Para contornar a restrição, injetámos uma tag de imagem com um caminho inválido acompanhado do manipulador de eventos `onerror`:

html

```
1 apple
```

O filtro falhou ao inspecionar este atributo e o navegador executou o código JavaScript com sucesso quando a imagem falhou o carregamento.

### 3.7.1 Testing for SQL Injection

Adicionalmente, explorámos o mesmo parâmetro de pesquisa recorrendo ao *sqlmap* para validar falhas de injeção SQL, conseguindo extrair com sucesso a estrutura de 22 tabelas da base de dados:

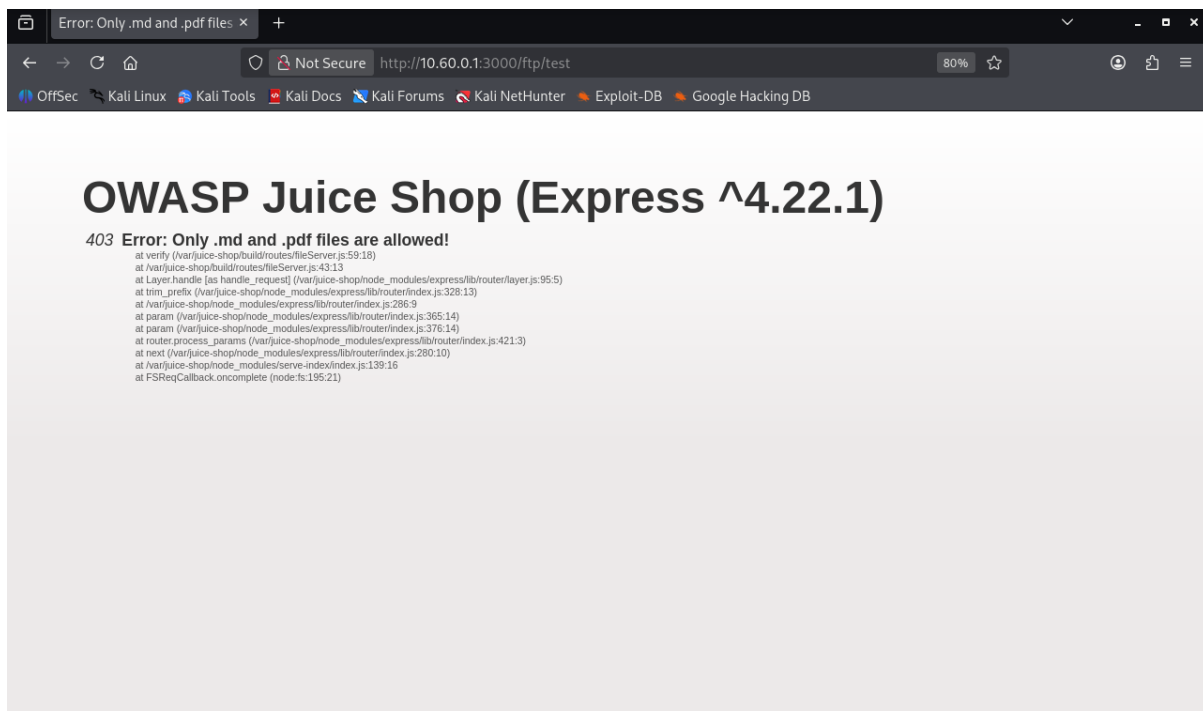
```
bash
1 sqlmap -u "http://10.60.0.1:3000/rest/products/search?q=apple" -p q --dbms
  =sqlite --prefix="'" --suffix="'" --tables --batch
2
3 [22 tables]
4 +-----+
5 | Addresses |
6 | BasketItems |
7 | Baskets |
8 | Captchas |
9 | Cards |
10 | ChallengeDependencies |
11 | Challenges |
12 | Complaints |
13 | Deliveries |
14 | Feedbacks |
15 | Hints |
16 | ImageCaptchas |
17 | Memories |
18 | PrivacyRequests |
19 | Products |
20 | Quantities |
21 | Recycles |
22 | SecurityAnswers |
23 | SecurityQuestions |
24 | Users |
25 | Wallets |
26 | sqlite_sequence |
27 +-----+
```

Apesar de não ter sido detetado pelo active scan foi feito fuzzing nos detalhes de login para saber se estava vulnerável a esse tipo de ataques visto que existia essa vulnerabilidade noutros parâmetros. Verificamos que de facto também estava vulnerável a SQL Injection, e que a resposta era a tabela com o

### 3.8 Testing for Error Handling

Ao tentar forçar o acesso a uma página ou ficheiro inexistente no servidor de ficheiros, como por exemplo na rota `/ftp/teste`, a aplicação falhou ao tratar a exceção de forma segura. Em vez de apresentar uma página de erro genérica (404), o servidor devolveu uma resposta detalhada expondo o *stack trace* completo do ambiente *Express.js*, revelando caminhos internos do sistema de ficheiros do servidor.





### 3.9 Client Side Testing

Validámos que o token de sessão (JWT) do utilizador autenticado está armazenado diretamente no `localStorage` do navegador. Uma vez que o `localStorage` não possui mecanismos de proteção equivalentes à flag `HttpOnly` dos cookies, qualquer script executado no contexto da página consegue ler estes dados.

Utilizando a falha de XSS identificada anteriormente na barra de pesquisas, injetámos o seguinte payload direcionado:

html

```
1 apple
```

A execução deste vetor permitiu extrair o conteúdo do token diretamente do armazenamento local da vítima. Isto prova que um atacante pode automatizar a exfiltração destas informações e assumir a identidade de qualquer utilizador afetado sem necessitar de saber as credenciais de acesso de forma persistente.

## **4 Web Application Security Firewall**

### **4.1 Information Gathering**

### **4.2 Configuration and Deployment Management Testing**

### **4.3 Identity Management Testing**

### **4.4 Authentication Testing**

### **4.5 Authorization Testing**

### **4.6 Session Management Testing**

### **4.7 Input Validation Testing**

### **4.8 Testing for Error Handling**

### **4.9 Client Side Testing**

## **5 Conclusions**