ADDETC – Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores

MEET – Mestrado Engenharia Eletrónica e Telecomunicações

MEIM - Mestrado Engenharia informática e multimédia

Cibersegurança

Exercício 2

Turma:

MEIM-11D, MEET-11D

Trabalho realizado por:

Carlos Costa Nº45231

Tiago Martins Nº45240

Miguel Távora Nº45102

Duarte Domingues Nº45140

Docente:

José Simão

Data: 2/12/2021

Índice

Índice ilustrações

1. Introdução

O exercício 2 de Cibersegurança tem como objetivo explorar vulnerabilidades de uma página web. A aplicação possui diversas vulnerabilidades deliberadamente realizadas de forma a explorar essas mesmas vulnerabilidades.

Inicialmente será realizada uma análise estática de forma verificar possíveis vulnerabilidades na implementação e que podem ser corrigidas.

De seguida será feita a análise de código, esta análise é feita quando o programa se encontra a correr e são forçados ataques durante a sua execução para ver se é resistente ou não aos ataques realizados. Para realizar os ataques será utilizado um proxy de ataque nomeadamente o OWASP ZAP, este é um software de segurança de aplicações web de código aberto.

2. Desenvolvimento

Análise estática

1 ª Alínea

Para realizar esta alínea foi adicionado o código fonte da aplicação Juice Shop. Isto foi possível através da ação de clonagem do GitHub como se pode observar na figura que se segue:

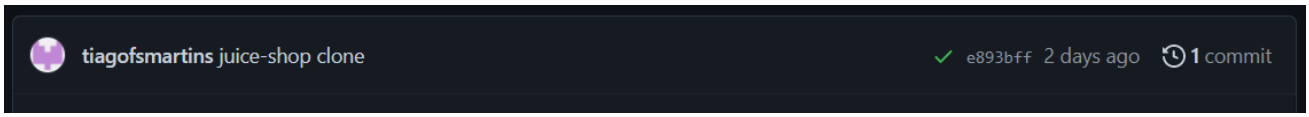

Figura 1 - Clonagem do código fonte do Juice Shop

2 ª Alínea

O GitHub permite ações de análise de segurança para o código fonte das aplicações. Para realizar esta análise é necessário ir para a parte de segurança e selecionar a opção de realizar scan de alertas. O resultado obtido foi o que se segue:

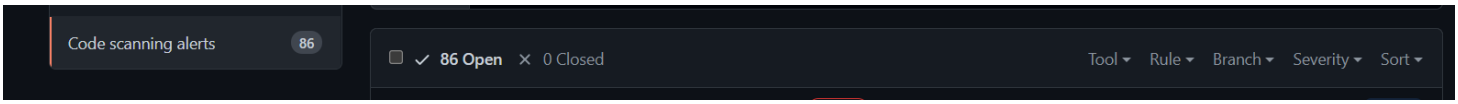

Figura 2 - Resultado da análise estática de código

3 ª Alínea

O CodeQL identificou a linha de código vulnerável porque no próprio código angular é feito uma *query* á base dados.

Isto é considerado uma vulnerabilidade porque quem deve fazer a *query* à base dados deve ser o servidor e não o próprio cliente, sendo o cliente a executar a *query* o cliente tem acesso ao código fonte e pode alterar esse código.

Com esta vulnerabilidade o cliente pode alterar a *query* feita á base dados e correr código malicioso, nomeadamente aceder a dados sensíveis, alterar dados, desligar a DBMS, aceder ao conteúdo de um ficheiro presente no sistema de ficheiros e emitir comandos ao sistema operativo.

A este tipo de ataque é designado SQL Injection, onde o cliente realizada uma consulta através dos dados de entrada do cliente para a aplicação. Neste caso a fonte é o cliente e o destino é a base dados.

4 ª Alínea

Um falso positivo é quando o sistema assume que é uma vulnerabilidade quando na verdade não existe nenhuma vulnerabilidade. Neste caso seria o sistema classificar um pedido do cliente diretamente à base dados, quando na verdade não é o cliente que realiza o pedido à base dados diretamente.

Um falso negativo é quando o sistema não deteta nenhuma vulnerabilidade, quando na realidade existe uma vulnerabilidade real no sistema. Isto poderia acontecer se fosse realizada uma *query* à base dados diretamente e esta devolvesse uma resposta à *query* do cliente.

5 ª Alínea

A análise realizada no ponto 3. foi uma análise feita em TypeScript. A análise foi realizada localmente isto porque a vulnerabilidade foi detetada relativamente aos argumentos de uma função em Angular. No caso de se tratar de uma vulnerabilidade global o código vulnerável seria um conjunto de funções e não somente os argumentos de uma função especificamente.

Análise dinâmica e *proxy* de ataque

6 ª Alínea

Foi alojada a aplicação na Cloud, usando uma VM do projeto GCP “CD2122D-G10”. Foi lhe atribuído o IP interno 10.132.0.4. É possível aceder à aplicação através de qualquer máquina pelo IP externo no porto 3000.

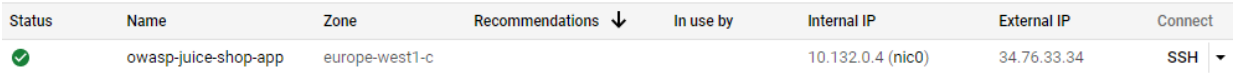

Figura 3 - VM a alojar a aplicação

7 ª Alínea

Com recurso às ferramentas de programador do browser foi descoberto o caminho que dá acesso à lista de classificações no ficheiro “main.js” presente na secção Sources. No conteúdo do ficheiro foi encontrado o caminho “score-board”, como se pode observar pela figura seguinte.

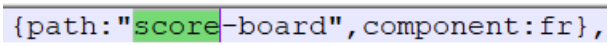

Figura 4 - Caminho da lista de classificações

Acedendo a esse caminho foi possível verificar a lista de classificações dos vários ataques possíveis a realizar na aplicação.

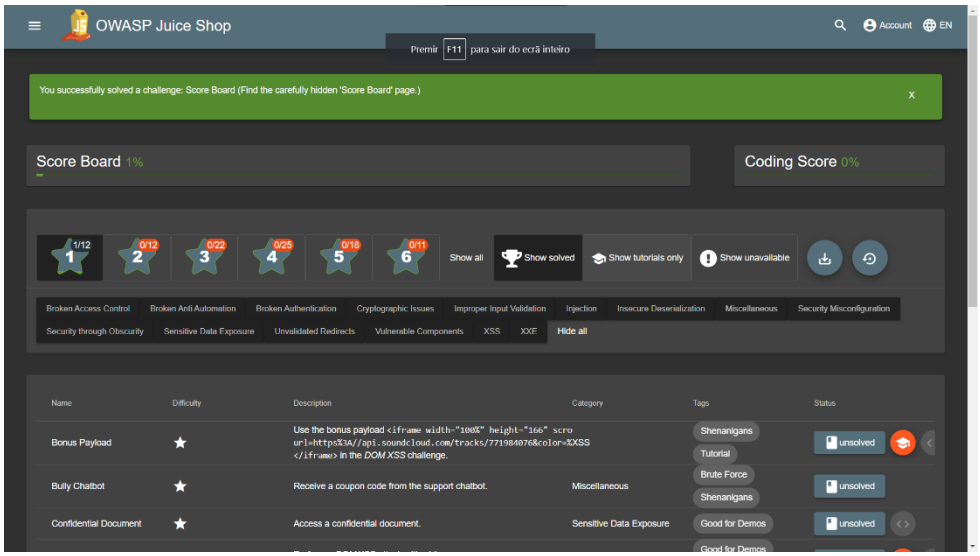

Figura 5 - Lista de classificações

8 ª Alínea

Neste ponto o objetivo é realizar o ataque de injeção de SQL sobre o formulário do login e para tal foram realizados os seguintes procedimentos:

1. Foi colocado o carater ‘ no campo do email e uma password aleatória de forma a verificar o comportamento de erro da aplicação. Através das ferramentas de programador do browser foi encontrado um erro na secção “Network” do tipo SQLITE_ERROR, onde foi possível encontrar o seguinte pedido SQL na resposta 500 a “/login”:

   SELECT * FROM Users WHERE email = '" AND password = '6c8349cc7260ae62e3b1396831a8398f' AND deletedAt IS NULL

2. Desta vez foi inserido no campo do email a string “’ OR TRUE” de forma a que o pedido SQL fique com o seguinte formato:

   SELECT * FROM Users WHERE email = '' OR TRUE' AND password = '6c8349cc7260ae62e3b1396831a8398f' AND deletedAt IS NULL

3. Como apenas nos interessa que o pedido SQL termine após a string “OR TRUE”, foi colocado todo o código seguinte em comentário através da inserção dos carateres “--” no final do email. Desta forma a query SQL resultante foi a seguinte:

   SELECT * FROM Users WHERE email = '' OR TRUE --' AND password = '6c8349cc7260ae62e3b1396831a8398f' AND deletedAt IS NULL

   Assim toda a secção a azul é ignorada e como o pedido SQL termina com a string “OR TRUE” a verificação do utilizador através do /login é sempre válida.

9 ª Alínea

Neste ponto foi instalada a ferramenta Zed Attack Proxy (ZAP) e foi verificado o seu correto funcionamento.

10 ª Alínea

Com recurso à ferramenta ZAP foi descoberta a password do utilizador administrador "admin@juice-sh.op", uma vez que se sabe à partida que a sua password começa por "admin" e tem um sufixo numérico de 3 algarismos. Para tal foi criado um fuzzer do tipo Numberzz que começa em 0 e termina em 999 de forma a realizar um ataque Fuzz ao pedido Post resultante da tentativa de login. No final deste ataque verificou-se que de entre todas as tentativas de login apenas uma possuia um body de 831bytes (ao contrário dos 26bytes das outras respostas).

Observou-se que o payload dessa mensagem é 123 logo a password correta do utilizador é "admin123".

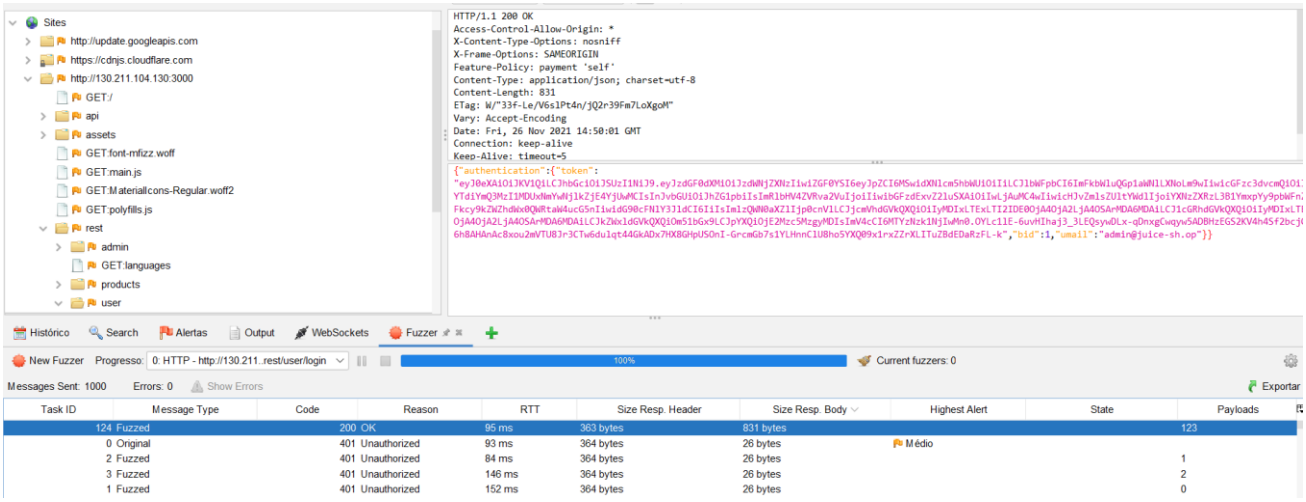

Figura 6 - Resultado do ataque Fuzz na ferramenta ZAP

11 ª Alínea

Neste ponto foram pesquisados produtos com a palavra "Lemon" na aplicação e verificou-se que no URL resultante está presente a palavra pesquisada como se pode verificar de seguida: "https://34.76.33.34:3000/#/search?q=Lemon". Assim concluiu-se que é possível realizar pesquisas através do próprio URL e experimentou-se através deste método pesquisar por "Lemon1", sendo apresentada esta palavra após a string "Search Result - " da página web como se pode verificar na imagem seguinte:

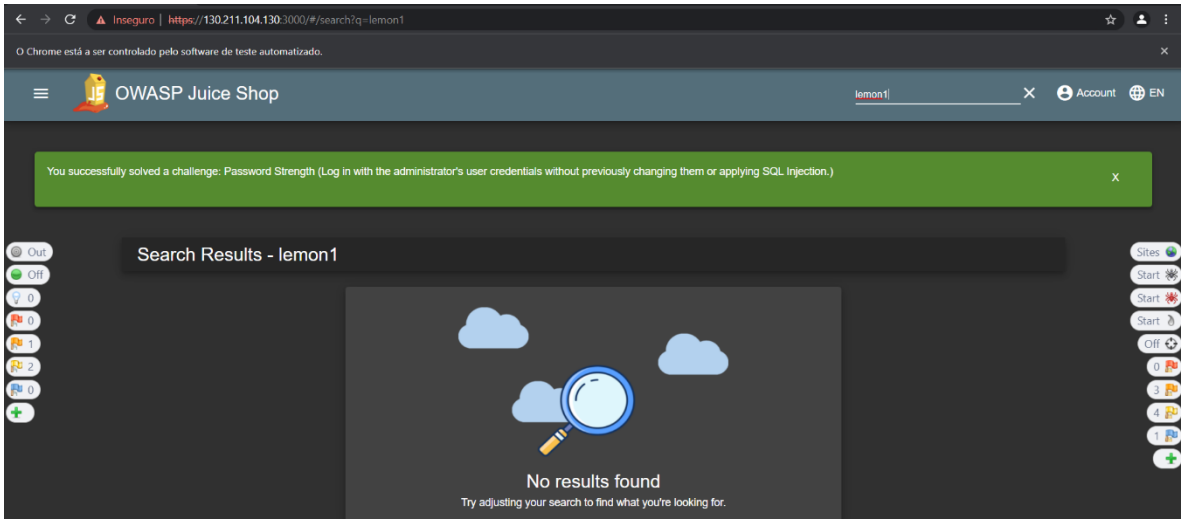

Figura 7 - Resultado da pesquisa Lemon1

12 ª Alínea

Neste ponto foi injetado o texto "<iframe src="javascript:alert(`xss`)">" de maneira a que o browser tenha de processar uma página com o texto injetado, tal como é pedido no desafio "DOM XSS". Para tal, foi utilizada a técnica utilizada no ponto anterior em que se colocou o texto pretendido no URL como se de uma pesquisa se tratasse. O que se verifica é que o browser em vez de interpretar o texto como uma palavra ele trata-o como código executável, e por essa razão o texto inserido não aparece nos resultados da pesquisa sendo por sua vez executado o código nele presente que neste caso resulta no surgimento de um alerta com a string "xss".

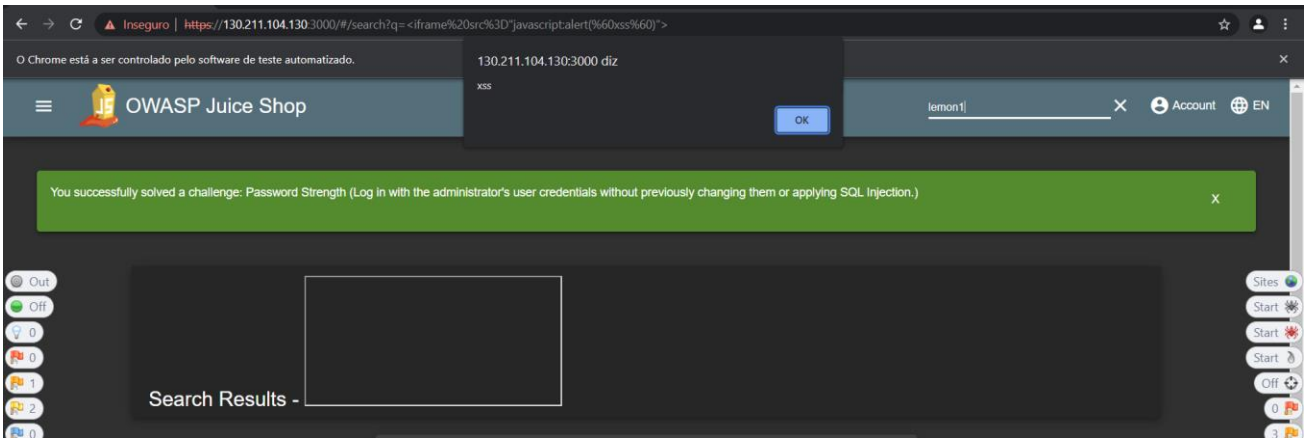

Figura 8 - Resultado do desafio DOM XSS

13 ª Alínea

Na secção Conctact da página web utilizada para efeitos de teste enviou-se o feedback através do proxy ZAP onde se alterou o HTTP request de forma ser possível submeter comentários de avaliação no nome de outros utilizadores. Para tal foi alterado o valor do campo “UserId” da mensagem.

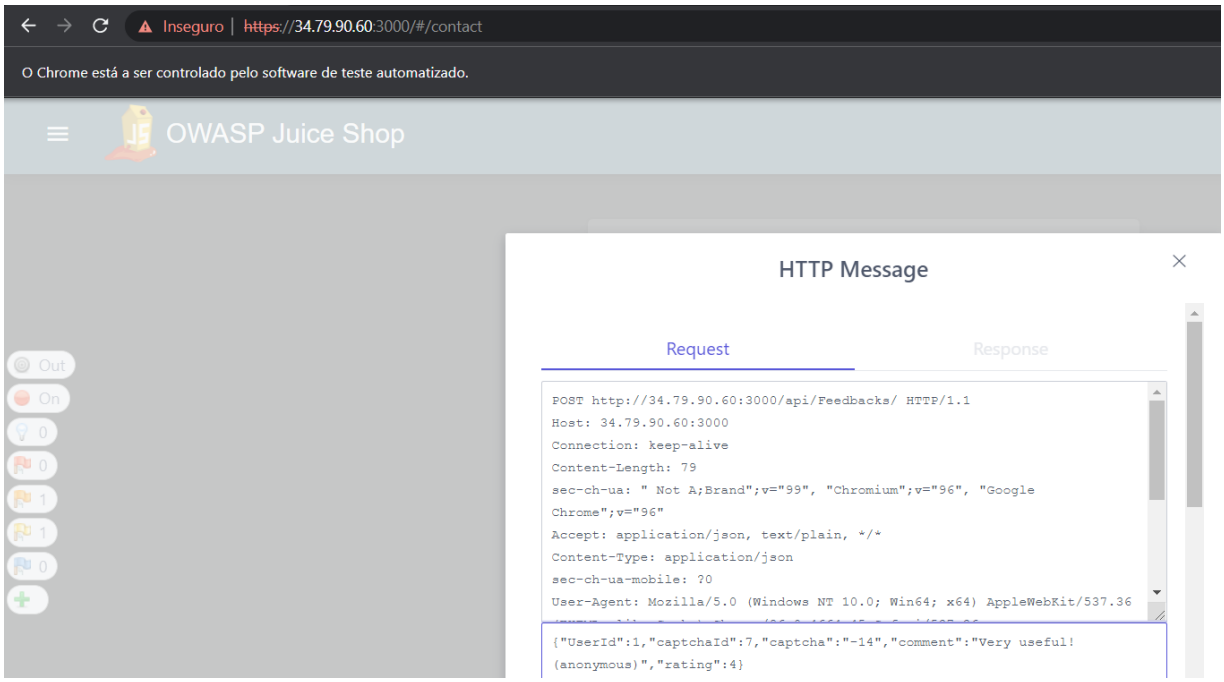

Figura 9 - Modificação da mensagem HTTP através do ZAP

Como se pode verificar, não tendo feito login na página foi possível criar um feedback associado ao UserId com o valor igual a 1, que é o Id do user admin@juice-sh.op.

Observou-se no local About do website o feedback que se criou em nome de outro utilizador como se pode observar na figura seguinte:

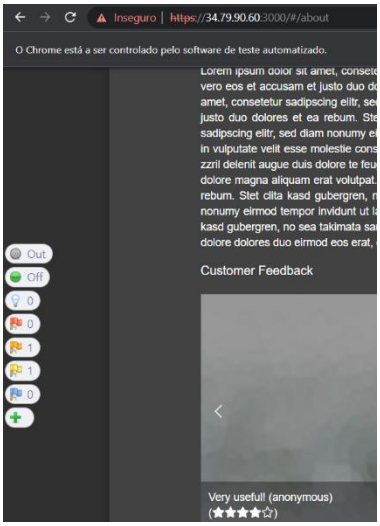

Figura 10 - Feedback gerado para outro utilizador

De seguida, foi também alterado o nome do user associado ao feedback criado através da modificação da mensagem http do campo comment.

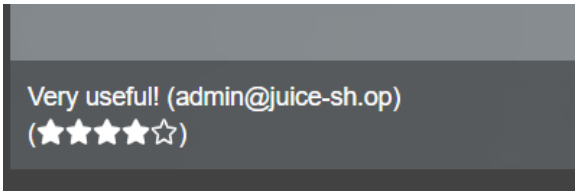

Figura 11 - Feedback gerado para outro utilizador após alteração do campo comment

14 ª Alínea

É possível obter o cookie com nome “token” armazenado no browser da vítima através de phising onde o atacante modifica o url do site e envia para a vítima, por exemplo, via email de forma a que a vítima no momento em que acede a esse url é realizado um ataque de XSS que tem como finalidade obter o cookie da mesma. O método para mostrar o valor da cookie com nome “token” é baseado no utilizado no ponto 12, onde é apresentado o valor de “document.cookie”.

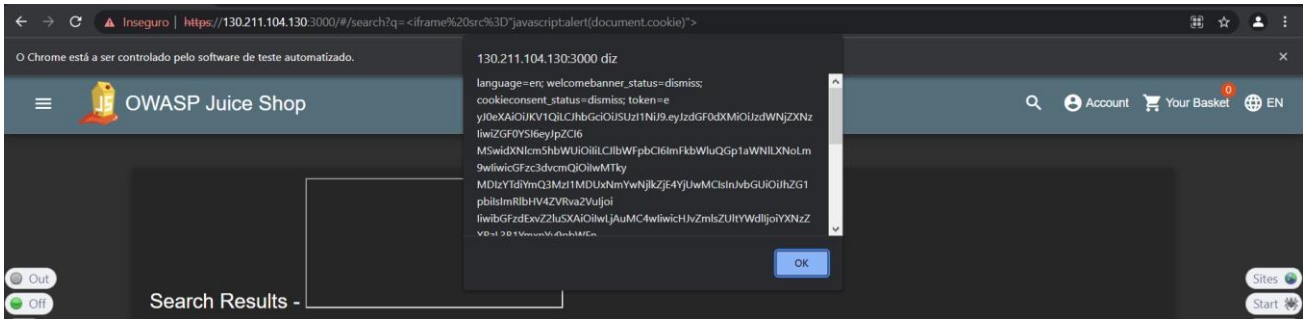

Figura 12 - Valor da cookie de nome "token"

3. Conclusões

Em suma, com este exercício foi possível explorar vulnerabilidades numa aplicação web. Inicialmente foram exploradas as funcionalidades da análise de vulnerabilidades de código realizadas pelo GitHub. Posteriormente foi posta uma aplicação web vulnerável a correr numa máquina virtual e foram explorados ataques possíveis de realizar a esta aplicação.