Instituto Superior de Engenharia de Lisboa

Cibersegurança, Módulo 2

**Trabalho Prático 2**

Mestrado em Engenharia Informática de Multimédia

Pedro Gonçalves, 45890

Rodrigo Dias, 45881

Rúben Santos, 49063

Semestre de Inverno, 2021/2022

**1)**

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto, monitor, interior, preto

Descrição gerada automaticamenteO repositório para o grupo **11** foi criado com sucesso.

**2)**

Foi gerado o seguinte ficheiro de ***workflow***:

Uma imagem com texto

Descrição gerada automaticamente

**3)**

Uma imagem com texto, captura de ecrã, monitor, interior

Descrição gerada automaticamenteA vulnerabilidade ***“Database query built from user-controlled sources”*** sobre o ficheiro **routes/login.ts** foi devidamente identificada pela **Github Action CodeQL**.

O **CodeQL** detetou esta linha de código como vulnerável porque existe uma potencial vulnerabilidade a injeções **SQL**. A ***query*****SQL** criada nesta linha de código será concatenada com dados obtidos diretamente do utilizador, sem que exista qualquer tipo de validação (***sanitization***) sobre os mesmos. A informação concatenada com uma ***query***em **SQL**, de forma a evitar injeções, deve ser sempre devidamente validada para que quaisquer valores inesperados sejam manuseados de forma correta e consistente.

A fonte (***source***) da vulnerabilidade são os dados inseridos pelo utilizador. O destino (***sink***) será a ***query*****SQL** efetuada em código que pode retornar potencial informação sensível.

**4)**

No contexto das injeções **SQL**, um exemplo de falso positivo identificado por uma ferramenta de análise de vulnerabilidades, seria uma ***query*****SQL** que utiliza dados inseridos pelo utilizador, apesar de terem sido devidamente e previamente validados. De modo a tratar-se realmente de um falso positivo, o código deve cobrir todas as rotas de ataque possíveis (***sinks***), de modo validar os dados de forma consistente.

Um exemplo de falso negativo seria uma ferramenta de análise de vulnerabilidades determinar que os dados inseridos pelo utilizador são devidamente validados, mas na realidade existem rotas de ataque (***sinks***) que podem ser exploradas.

**5)**

Dado que na função onde a ***query*****SQL** está explicitada, antes da linha de código crítica (onde é efetuada a ***query***), é chamada a função **verifyPreLoginChallenges*()***que, pelo nome e parâmetros que aceita, deveria realizar a validação dos dados inseridos pelo utilizador, conclui-se que a análise que identificou esta vulnerabilidade é de contexto local. Deste ponto de vista, talvez a vulnerabilidade encontrada até se trate de um falso positivo, no caso de a validação dos dados por parte da função **verifyPreLoginChallenges*()***ser feita de forma correta, consistente e cubra todas as rotas de ataque possíveis.

**6)**

A aplicação WEB encontra-se hospedada no projeto da Google Cloud Platform **CD2122D-G11** na máquina virtual com **IP 34.142.123.208** e utiliza o porto **3000**.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente**7)**

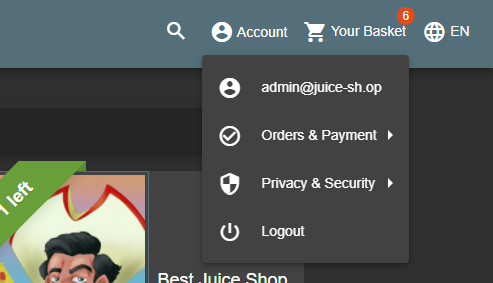
Tela de celular com jogo

Descrição gerada automaticamenteO desafio foi completado. Verifica-se que aceder a uma página secreta de uma aplicação WEB vulnerável é extremamente fácil. Com recurso ao ficheiro javascript que contém os váriso caminhos da aplicação, facilmente se descobre o nome das páginas secretas, incluindo a do score-board, bastando utilizar o **URL** **http://34.142.123.208:3000/#/score-board**

**8)**

De forma a manipular a ***query*** **SQL** interna realizada pela aplicação **WEB**, digitou-se o seguinte:

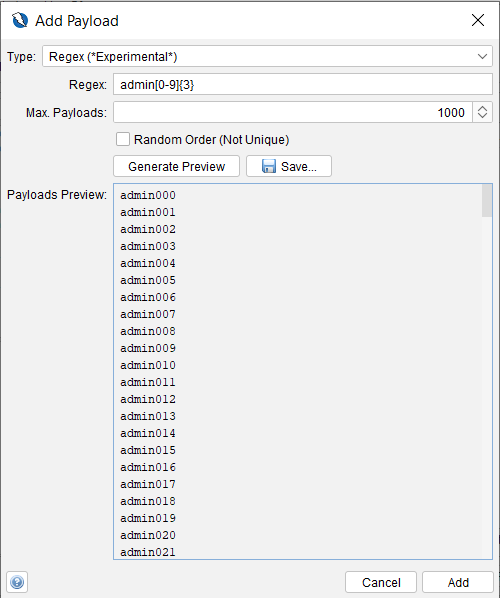
**‘ OR TRUE--**

Interface gráfica do usuário

Descrição gerada automaticamenteEsta sucessão de caracteres irá validar a autenticação sem sequer olhar para o email ou para a password.

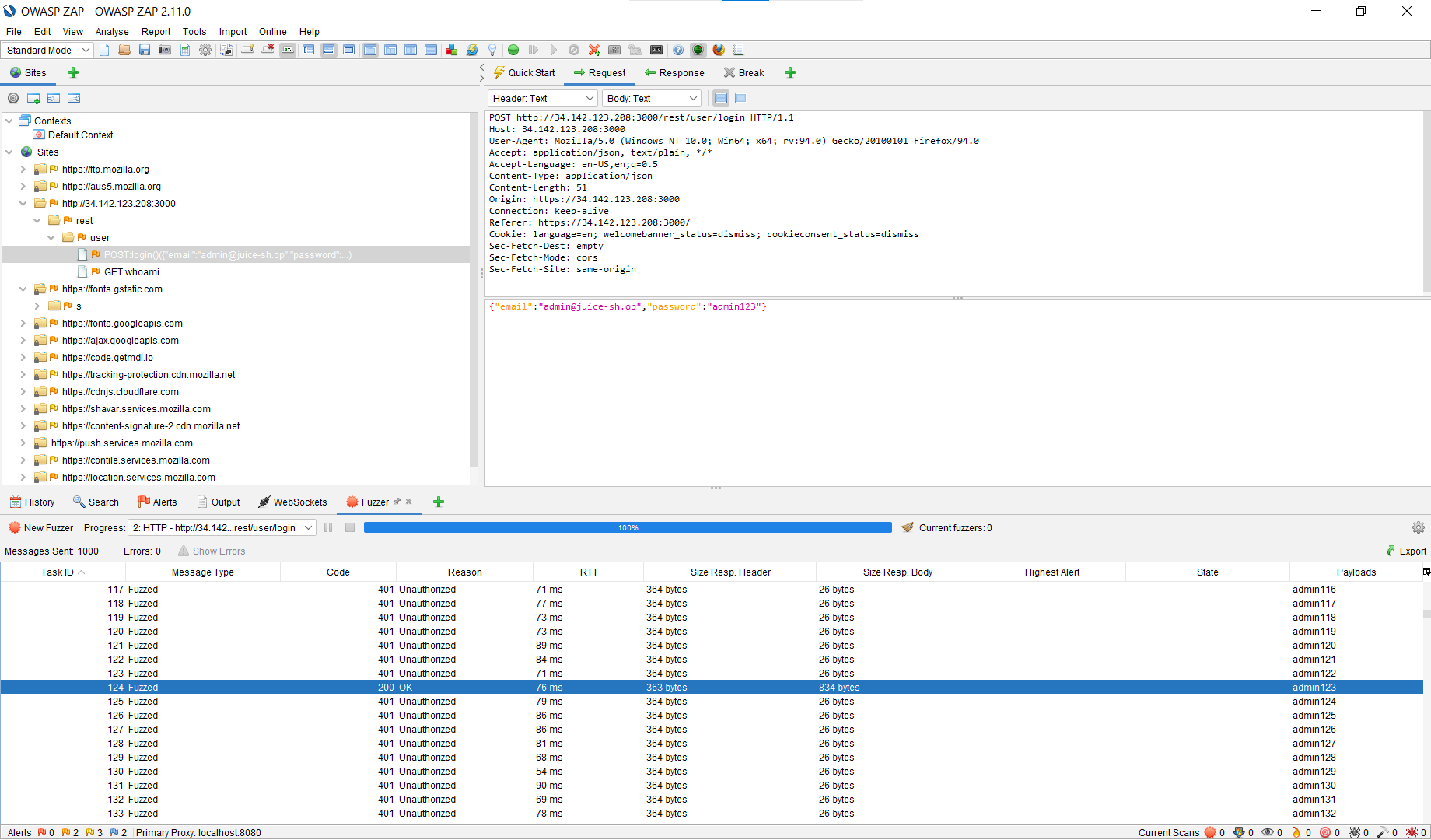
Interface gráfica do usuário

Descrição gerada automaticamente**9)**

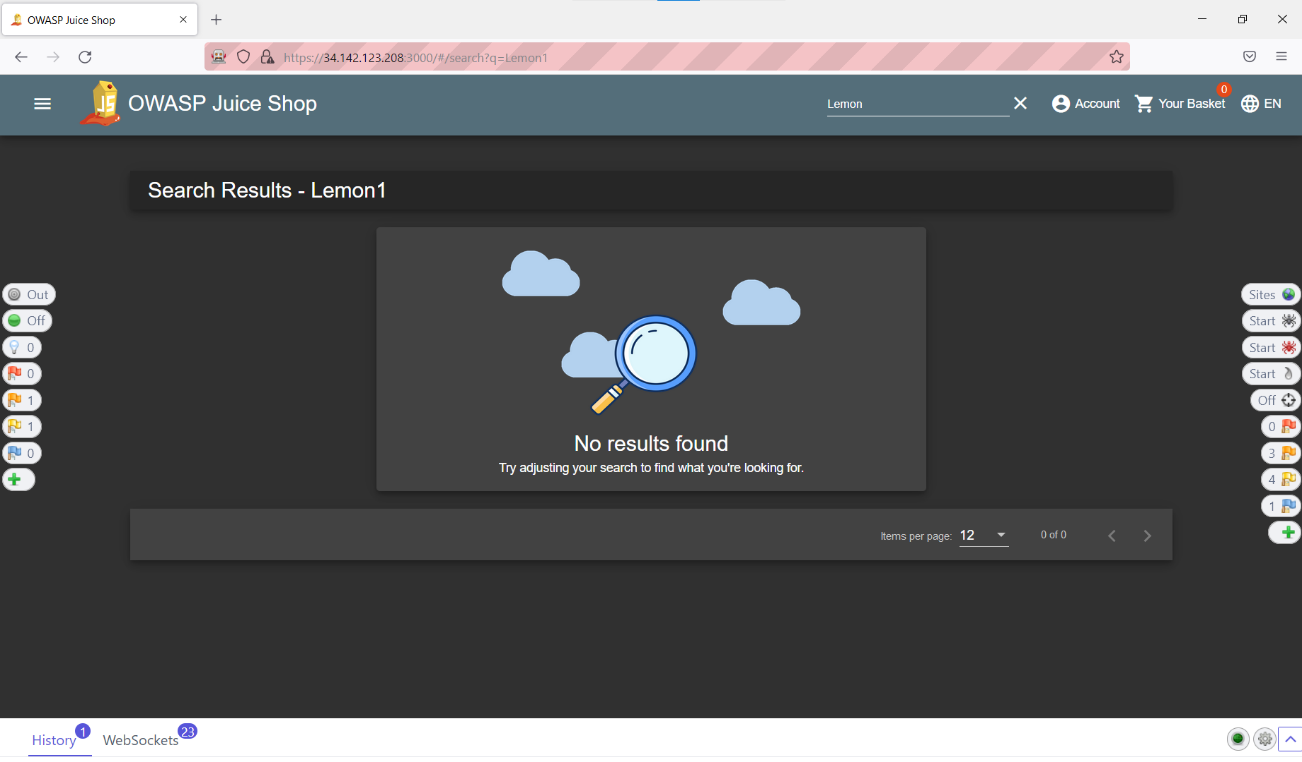
**10)**

Ao introduzir credenciais incorretas no menu de ***login***, o software **ZAP** apanha uma request da aplicação **WEB** no servidor. Interpretando, verificam-se, em formato **JSON**, os parâmetros introduzidos. Associado ao parâmetro password, existem algumas configurações relacionadas com **Fuzzing**, às quais se podem adicionar ***payloads***.

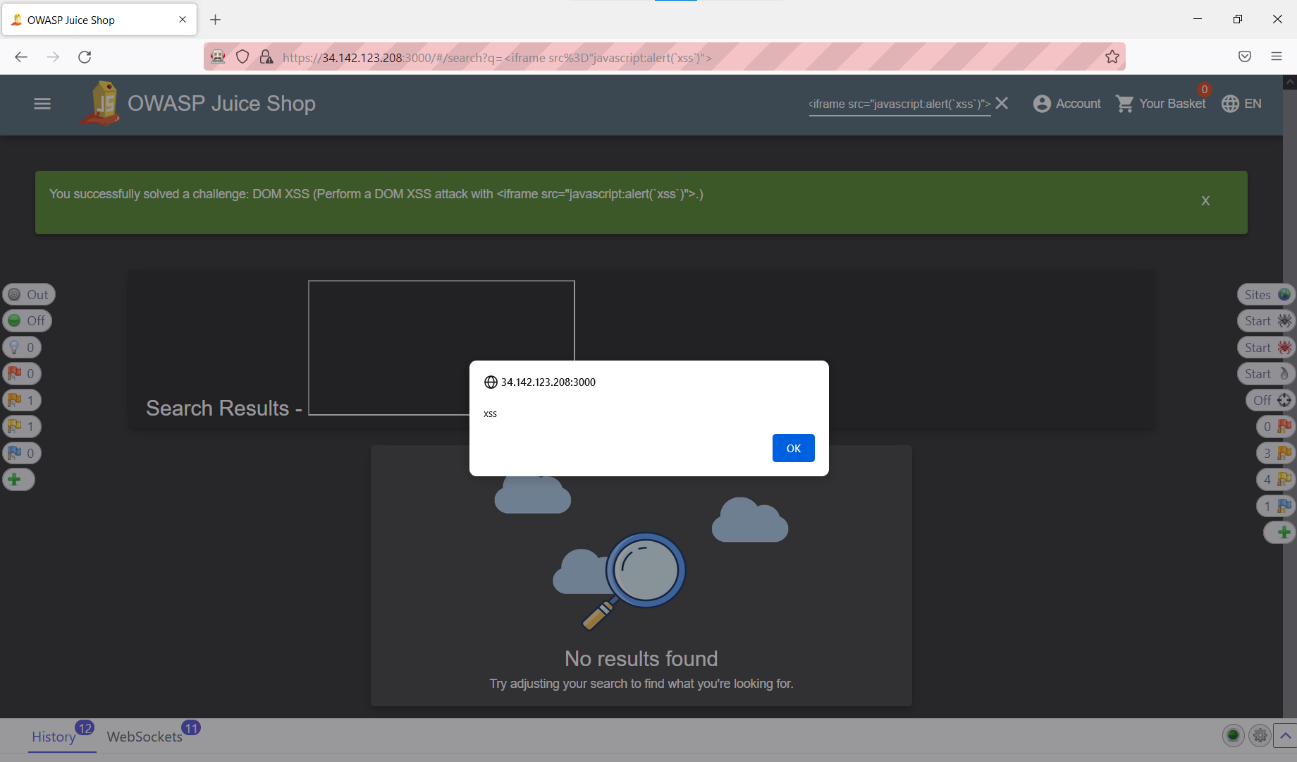
Ao introduzir potenciais passwords (de acordo com as restrições indicadas pelo enunciado), o tipo de ataques a realizar (**SQL** ***Injections***) e ao iniciar o **Fuzzer**, verifica-se que o **ZAP** procede à tentativa de autenticação com as passwords introduzidas, tendo, no caso da tarefa **124**, obtido uma autenticação bem sucedida.



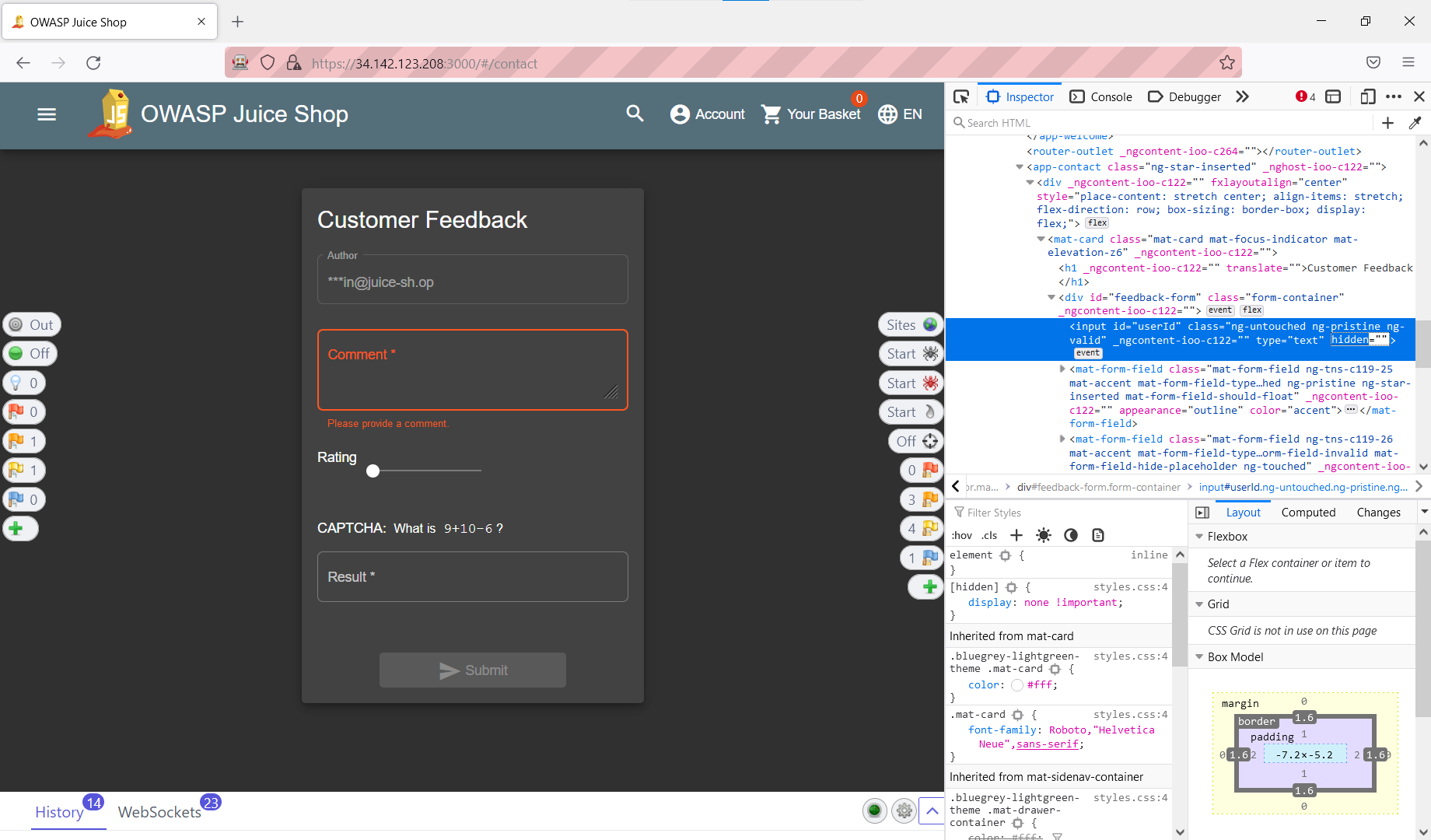
**Interface gráfica do usuário, Aplicativo

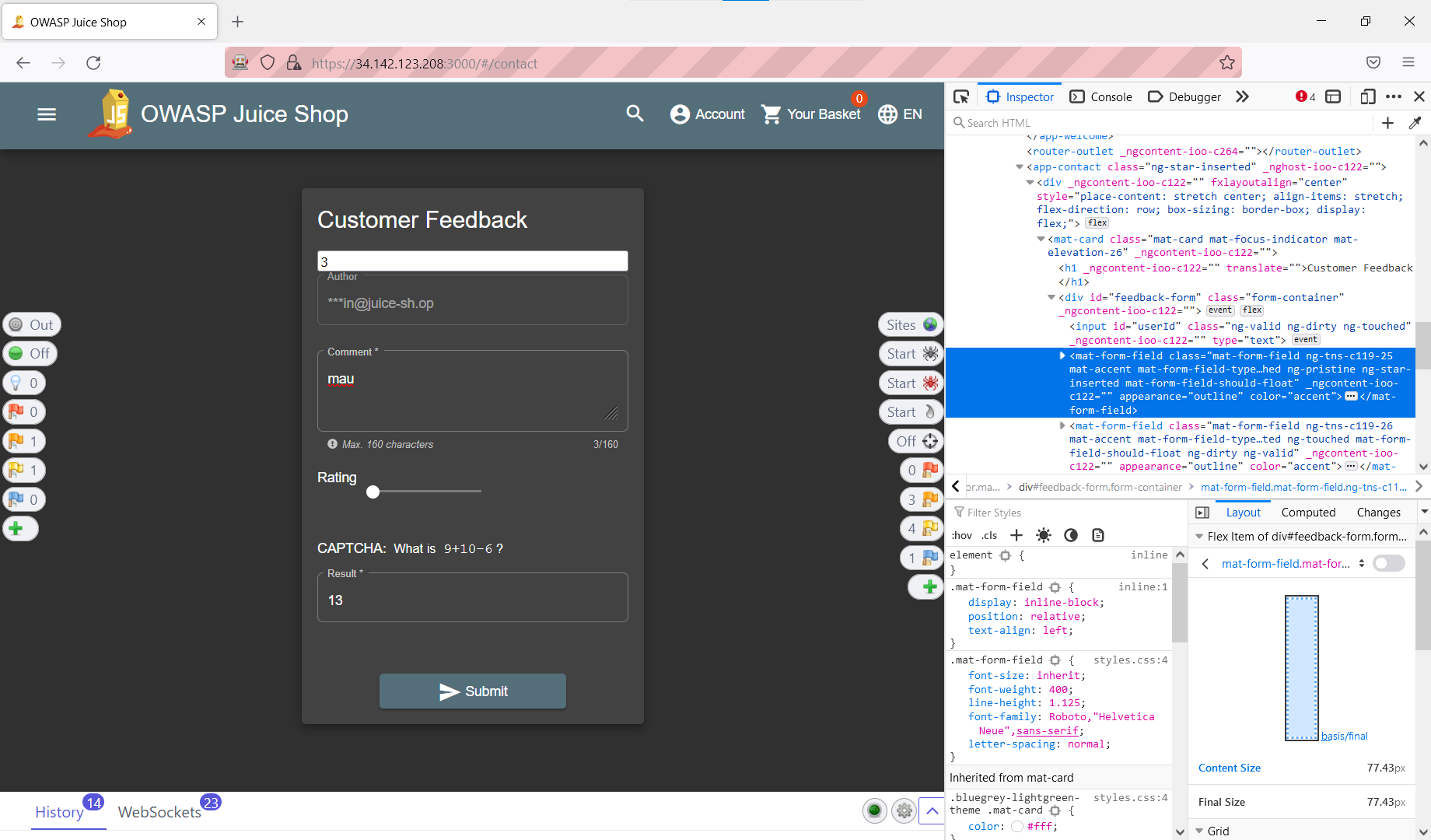
Descrição gerada automaticamente****11)**

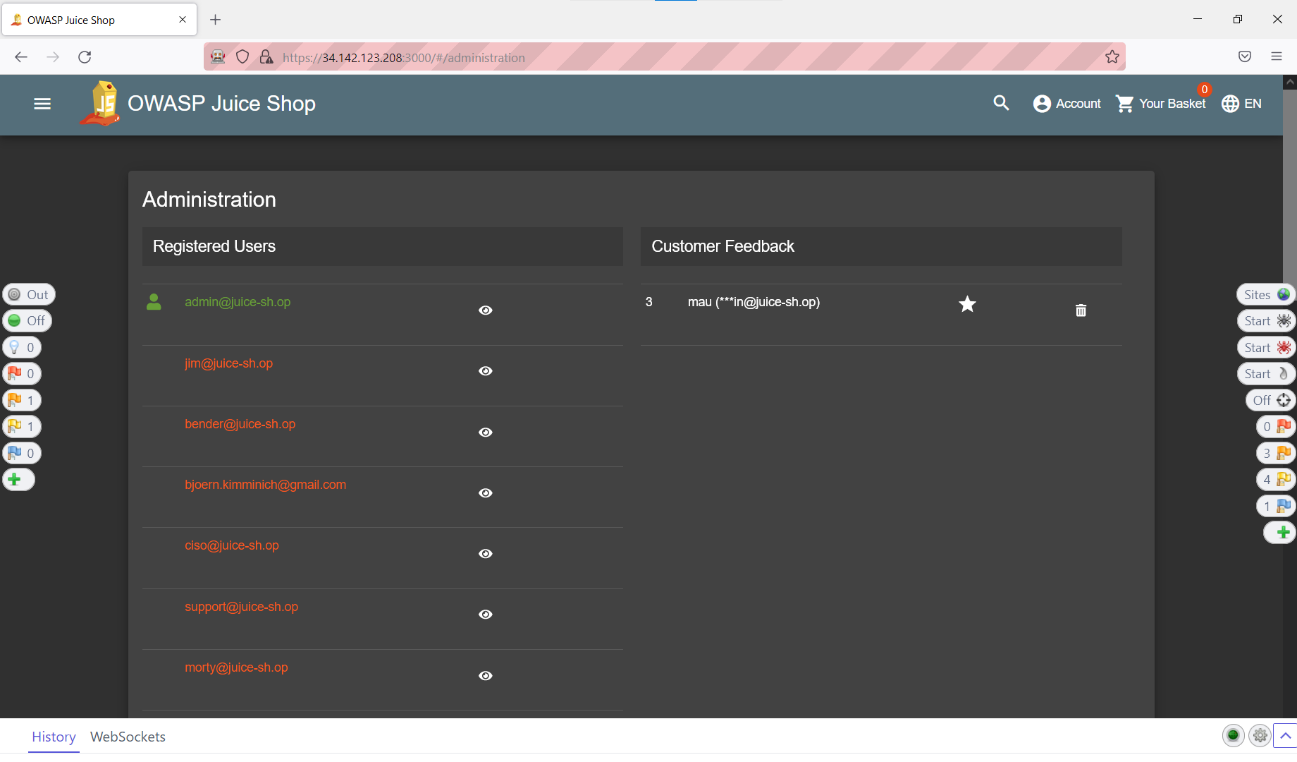
**12)**

****Ao introduzir **“<iframe src=”javascript:alert(‘xss’)”>”** na barra de procura, na verdade está-se a injetar código javascript para a aplicação **WEB**, de forma a mostrar um ***Alert******Dialog*** com a **String** “xss”.

**13)**

Ao explorar o **Feedback** **Form** da página de ***feedback***, verfica-se que existe um campo de texto escondido que corresponde ao **ID** do usuário (**UserId**) autor do ***feedback*** a publicar.

****Ao manipular o parâmetro **“hidden”** associado a esse campo de texto, tornando-o visível, é possível falsificar a autoria do comentário de ***feedback***.

****Vistando a página “administration” verifica-se que o comentário foi publicado em nome do utilizador **3**.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamentePara explorar esta vulnerabilidade com recurso ao ***software*** **ZAP**, utilizar-se-á uam funcionalidade que permite intercetar ***requests***. Ao submeter um comentário de feedback na autoria da conta admin, o **ZAP** interceta o seguinte request:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamenteVerifica-se que, em formato **JSON**, o **ZAP** interceta, juntamente com os restantes parâmetros do **Feedback Form**, o parâmetro **UserId**. Alterando-o para, por exemplo, **7**, o **ZAP** interceta imediatamente outro ***request***:

Tela de computador com jogo

Descrição gerada automaticamenteEste request provém da completação do desafio. Visitando novamente a página **“administration”**, verfica-se que o comentário foi publicado em nome do utilizador **7**.

**14)**

Para obter a cookie de nome ***“token”*** teremos de recorrer novamente a um ataque ***cross-site*** ***scripting*** que nos permite visualizar as cookies armazenadas na página **WEB** em questão. De forma a obter todas as cookies armazenadas, em javascript, utiliza-se o seguinte código:

**document.cookie**

Este código retorna uma String do tipo **“cookie1=value; cookie2=value...”**.

Assim, para visualizar todas as cookies numa caixa de diálogo de alerta, o seguitne código será introduzido na barra de pesquisa:

**Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente<iframe src="javascript:alert(document.cookie)">**