Cibersegurança – Módulo 2

Segundo trabalho prático

Pressupostos:
O grupo tem acesso a:
- Um projeto na Google Cloud Platform (GCP)
- Um repositório GitHub dentro da organização "isel-deetc-computersecurity"

Entrega:
Relatório com evidências e/ou explicações pedidas em cada questão, o nome da máquina virtual criada no GCP e o porto usado para alojar a aplicação Juice Shop.

O objetivo é fazer análise de código estático e análise dinâmica no contexto da aplicação web Juice Shop. Esta aplicação tem vários erros de programação deliberados para que os utilizadores deste sistema possam obter maior conhecimentos sobre vulnerabilidades em geral e em aplicações web em particular.

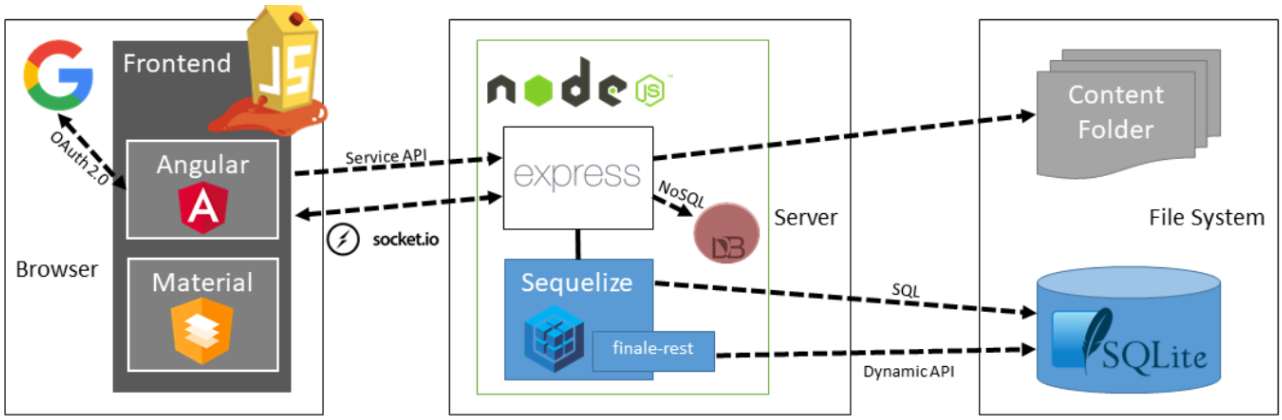

Figura 1: Arquitetura da Juice Shop

A arquitetura da *Juice Shop* inclui um *frontend* desenvolvido em Angular, um *backend* desenvolvido em node.JS e o uso de base de dados relacionais e não relacionais, como ilustrado na Figura 1.

Análise estática

1. Obtenha o código fonte da aplicação Juice Shop, https://github.com/juice-shop/juice-shop e adicione o código ao repositório GitHub do seu grupo.

   Foi enviada uma mensagem pelo Moodle com um *link* para criação de um repositório público dentro da organização "isel-deetc-computersecurity".

2. Configure uma GitHub Action para executar, num evento de *push*, as análises CWE da plataforma CodeQL no repositório criado no ponto anterior

3. Considere a vulnerabilidade CWE-89 "*Improper Neutralization of Special Elements used in an SQL Command*" (https://cwe.mitre.org/data/definitions/89.html). Procure a entrada *"Database query built from user-controlled sources"* sobre o ficheiro `routes/login.ts` na lista de vulnerabilidades detectadas pela Github Action CodeQL. Justifique porque motivo o CodeQL identificou este código como vulnerável. Inclua informação sobre a fonte (*source*) e o destino (*sink*). (https://owasp.org/www-community/attacks/SQL_Injection).

4. A análise estática de código pode ter erros, nomeadamente falsos positivos ou falsos negativos. No contexto do exemplo anterior, onde foi detetada uma possível vulnerabilidade de Injeção de SQL, dê exemplos do que poderia ser um falso positivo e um falso negativo.

5. Considere a seguinte documentação sobre o CodeQL e a capacidade de fazer análise de fluxo no código fonte, em Java e em javascript (https://codeql.github.com/docs/codeql-language-guides/analyzing-data-flow-in-java/#analyzing-data-flow-in-java) (https://codeql.github.com/docs/codeql-language-guides/analyzing-data-flow-in-javascript-and-typescript/#analyzing-data-flow-in-javascript-and-typescript). A análise realizada pelo CodeQL pode ser apenas local ou global. A interrogação QL que detetou a vulnerabilidade da ponto (3) faz qual deste tipo de análises?

Análise dinâmica e *proxy* de ataque

6. Aloje a aplicação na Cloud, usando uma VM de um projecto GCP, como explicado aqui: https://github.com/juice-shop/juice-shop#google-compute-engine-instance

7. Com as ferramentas de programador do browser (F12 no *chrome* ou *firefox*) descubra o caminho que dá acesso à lista de classificações - "score board", presente num dos ficheiros *javascript* carregados pelo *frontend*. Apresente as ações realizadas. Esta questão corresponde ao desafio "*Find the carefully hidden 'Score Board' page*": https://pwning.owasp-juice.shop/part2/score-board.html

8. Realize o ataque de Injeção de SQL sobre o formulário de Login, tendo como alvo o utilizador `admin`. Ajuda tutorial disponível neste link: https://demo.owasp-juice.shop/#/hacking-instructor?challenge=Login%20Admin.

9. Instale e verifique o correto funcionamento do Zed Attack Proxy (ZAP) disponível aqui https://owasp.org/www-project-zap/:
    a. Na opção "Quick start" escolha a opção "Manual Explore", e indique o site https://www.example.org, com HUD ativo, escolhendo o browser Firefox (pode ser necessário instalar o *browser*), e verifique consegue aceder ao site e aos comandos do ZAP na mesma janela.
    b. Aceda à aplicação Juice Shop através do browser lançado no ponto anterior.

10. Usando a ferramenta ZAP, descubra a *password* do utilizador administrador `admin@juice-sh.op`. A password começa por "admin" e tem um sufixo numérico de 3 algarismos. Mostre como pode através de fuzzing descobrir a *password* correta.

11. Procure por produtos com a palavra "Lemon". Repare na barra de endereços. Experimente alterar diretamente na barra de endereços o critério de pesquisa para "Lemon1". Na página de resultados onde é inserido o texto do critério de pesquisa?

12. Resolva o desafio "DOM XSS", injetando o texto `<iframe src="javascript:alert(`xss`)">` de maneira a que *browser* tenha de processar uma página com o texto injetado (https://pwning.owasp-juice.shop/part2/xss.html#perform-a-dom-xss-attack).

13. [extra] Mostre como resolver o desafio "*Post some feedback in another users name*" usando o proxy ZAP.

14. [*extra*] Mostre como, através de um ataque Cross-site scripting (XSS) e de engenharia social, um atacante pode obter o *cookie* com nome "`token`" armazenado no browser da vítima.