Cibersegurança - Segurança em Software (Módulo 2)
Trabalho Prático - Parte 2

Objetivos:
- Usar a ferramenta CodeQL para realizar análise estática de código (SAST), em ambiente local e em repositórios GitHub
- Usar a ferramenta ZAP Proxy para realizar análise dinâmica (DAST) em aplicações *web*

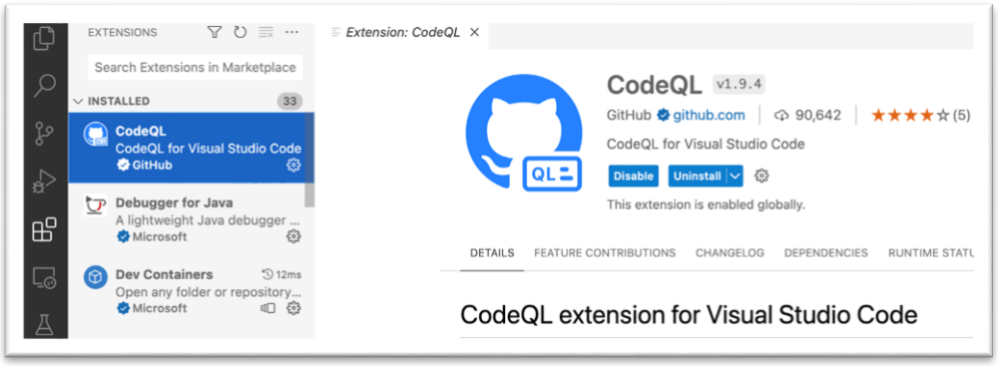
Entrega:
- Respostas e evidências solicitadas no enunciado (A.4, C.2, C.3, C.4, D.3 a D2.8)

A. CodeQL – Execução local [20 pontos]

1. Instale localmente as ferramentas da plataforma CodeQL através da aplicação Visual Code:

   a. Instale a aplicação Visual Code: https://code.visualstudio.com

   b. Instale a versão mais recente da extensão CodeQL do Visual Code.



   Passará a ter este icon:

   c. Instale as ferramentas da linha de comandos do sistema CodeQL, escolhendo a versão correta para o seu sistema operativo: https://github.com/github/codeql-cli-binaries/releases.

   d. Faça clone do *starter kit* para uma diretoria no seu computador: https://github.com/github/vscode-codeql-starter.

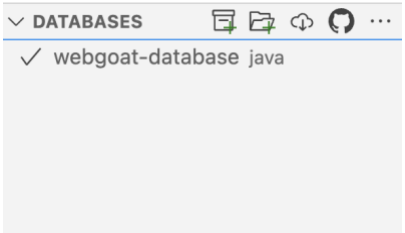
   e. Faça clone do repositório da aplicação de aprendizagem WebGoat: https://github.com/WebGoat/WebGoat. A aplicação WebGoat é uma aplicação web

deliberadamente insegura, para uso em contexto de treino sobre segurança no software.

2. Crie a base de dados CodeQL a partir do código fonte da WebGoat (comando executado na diretoria de base do código fonte). Tenha em conta que este comando irá tentar compilar a aplicação WebGoat. Para tal, precisa de ter instalado no computador uma JVM compatível com a versão Java (<java.version>) indicada no ficheiro pom.xml.

```
codeql database create webgoat-database --language=java --overwrite
```

3. Abra o *workspace* do *starter kit* no Visual Code e carregue a base de dados para o CodeQL no Visual Code:



4. Pretende-se avaliar vulnerabilidades do tipo *Use of Hard-coded Credentials* – CWE 798 (https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password, https://cwe.mitre.org/data/definitions/798.html):

    a. Encontre a interrogação QL correspondente no *workspace* do *starter kit* (`HardcodedPasswordField.ql`) e execute a interrogação.

    b. Na lista de resultados, escolha a segunda entrada referente ao ficheiro `JWTRefreshEndpoint.java`. Analise o código e descreva em que consiste a estrutura JWT (https://jwt.io), onde é usada uma das *password*s constantes, e qual a operação criptográfica que depende da *password*.

    c. No caso do JWT ser usado para verificar a identidade e características do utilizador, que tipo de ações o atacante pode explorar se conhecer a *password* escrita no código fonte?

B. Aplicação Web JuiceShop

No restante enunciado o objetivo é usar técnicas de análise estática e dinâmica direcionadas para a aplicação *web* Juice Shop. Esta aplicação tem erros de programação propositados que podem ser usados para obter conhecimento sobre vulnerabilidades em geral e aplicações web em particular.

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

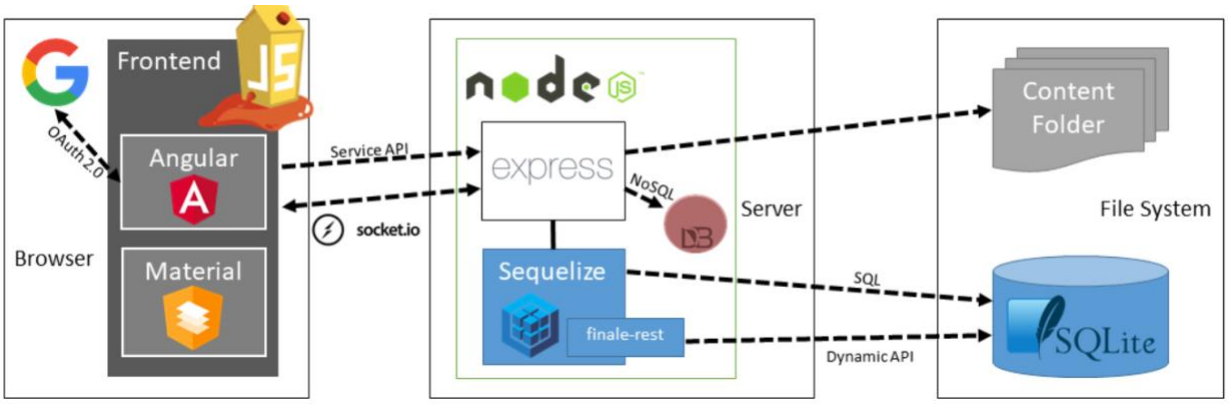


Figura 1: Arquitetura da Aplicação Juice Shop

Adicione o código-fonte da aplicação Juice Shop, https://github.com/juice-shop/juice-shop, ao repositório GitHub do seu grupo. Na página Moodle da disciplina há um *link* que deve seguir para criar o repositório do grupo através da aplicação Github classroom.

C. CodeQL – execução no Github [20 pontos]

1. Analise a ação do GitHub para executar a análise CWE do CodeQL, a qual é executada quando há um evento *push* no repositório. Altere o gatilho do fluxo de trabalho para ser acionado manualmente. (https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows#workflow_dispatch).

2. Nos *logs* de saída, após executar a ação, identifique a etapa e o comando usado pela Ação CodeQL para inicializar a base de dados com metadados sobre o código fonte. Onde fica a base de dados que resulta da análise do código?

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*" no arquivo `routes/search.ts` na lista de vulnerabilidades detetadas pelo Github Action CodeQL. Justifique o motivo do CodeQL identificar esse código como vulnerável. Inclua *informações de source* e *sink*. (mais

informações sobre SQL Injection podem ser encontradas aqui https://owasp.org/www-community/attacks/SQL_Injection).

4. A análise de código estático pode ter erros, designados como falsos positivos ou falsos negativos. Explique a diferença entre estas duas situações usando um exemplo.

D. Análise dinâmica em aplicações web e a ferramenta Zed Attack Proxy (ZAP) [60 pontos]

1. Execute a aplicação *JuiceShop* localmente no seu computador, usando um contentor Docker, de acordo com as instruções em https://github.com/juice-shop/juice-shop#docker-container.

2. Pode consultar todos os desafios que existem na aplicação acedendo ao URL http://localhost:3000/#/score-board.

3. Descreva e execute um ataque de injeção de SQL no formulário de *login*, tendo como alvo o utilizador administrador:
https://pwning.owasp-juice.shop/companion-guide/latest/part2/injection.html#_log_in_with_the_administrators_user_account

4. Instale e verifique o funcionamento correto do Zed Attack Proxy (ZAP) disponível aqui https://www.zaproxy.org:

    a. Na opção "Início rápido", escolha a opção "Manual Explore" e indique o site https://www.example.org, com o HUD ativo, escolhendo o navegador Firefox (pode ser necessário instalar o navegador). Verifique se pode aceder ao site e aos comandos ZAP na mesma janela.

    b. Aceda à aplicação Juice Shop através do *browser* lançado no ponto anterior.

5. Usando a ferramenta ZAP, descubra a senha do utilizador administrador `admin@juice-sh.op`. A senha começa com "admin" e termina num número com 3 algarismos. Mostre como usar a ferramenta de *fuzzing* para encontrar a senha correta.

6. Usando a ferramenta ZAP, Mostre como resolver o desafio "*Post some feedback in another user's name*" :

    https://pwning.owasp-juice.shop/companion-guide/latest/part2/broken-access-control.html#_post_some_feedback_in_another_users_name

7. Considere as vulnerabilidades do tipo Cross-site scripting (XSS):

    a. Procure produtos com a palavra "Lemon". Observe a barra de endereço. Tente alterar os critérios de pesquisa para "Lemon1" diretamente na barra de endereço. Na página de resultados, qual elemento HTML é usado para apresentar o texto dos critérios de pesquisa?

    b. Resolva o desafio "*DOM XSS*" injetando o texto `<iframe src="javascript:alert('xss')">` para que o navegador tenha que processar uma página com o texto injetado (https://pwning.owasp-juice.shop/companion-guide/latest/appendix/solutions.html#_perform_a_dom_xss_attack).

    c. Experimente injectar o script `<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076"></iframe>`.

    d. O servidor devia usar HTML seguro na geração da página de resposta à pesquisa. Existe um controlo de segurança designado de Content Security Policy (CSP) (ver por exemplo https://auth0.com/blog/defending-against-xss-with-csp/) que deve ser usado para maior proteção contra ataques de XSS. Descreva sucintamente como é este mecanismo poderia ser usado para evitar os ataques anteriores mas mesmo assim permitir à aplicação web o uso de javascript *inline*.

8. Considere a vulnerabilidade de *cross-site scripting* (XSS) explorada no ponto 7.b:

    a. Descreva como é que usando engenharia social, um invasor poderia obter o *cookie* de nome "token", armazenado no navegador da vítima. Não é preciso executar cada uma das etapas, mas deve incluir uma explicação convincente do processo e descrever o que pode ser obtido do "token".

    b. Descreva um possível vetor de ataque usando os fatores “vulneratibility factors” e “tecnhical impact factors” da metodologia OWASP para cálculo de risco (https://javierolmedo.github.io/OWASP-Calculator/).