

Engenharia de Software Seguro

Laboratório 4 — Vulnerable Apps (Mobile + Web)

Grupo 6

Elementos: Hugo Nunes, Miguel Teixeira

Repositório GitHub: [c-academy-ess/api-todo-list-manager-grupo-6](https://github.com/c-academy-ess/api-todo-list-manager-grupo-6)

Tag(s): lab4

Data: 14-02-2026

Objetivo. Identificar e explorar vulnerabilidades em aplicações móveis e web usando análise estática e dinâmica, conforme enunciado do Laboratório 4.

Ferramentas. Bytecode Viewer, Java/JCE, Docker, OWASP ZAP, Visual Studio Code + extensão CodeQL.

Conteúdo

1. Preparação de ferramentas.....	3
2. Ataques em aplicações móveis (Lab4-ess.apk).....	3
2.1 Artefactos encontrados (frase cifrada, chave e algoritmo).....	3
2.2 Resolução com DecodeAndroidKey.java	4
3. CodeQL — análise local (vulnapp-ess.zip + vulnapp_db.zip).....	5
3.1 CWE-79 (XSS).....	5
3.2 CWE-89 (SQLi)	5
3.3 CWE-327 (cripto fraca).....	6
4. Análise dinâmica — OWASP Juice Shop.....	6
4.1 Execução em Docker.....	6
4.2 Descobrir o Score Board (rota escondida)	7
4.3 SQL Injection no login (admin)	7
4.4 ZAP — Manual Explore	9
4.5 Fuzzing da password admin###	9
4.6 Alterar termo de pesquisa na barra de endereço.....	11
4.7 DOM XSS	11
4.8 Persisted XSS via API.....	12
4.9 Feedback em nome de outro utilizador	12
4.10 Roubo do cookie/token.....	13
5. Conclusão	13
Referências	14

1. Preparação de ferramentas

- Bytecode Viewer (v2.13.2) para descompilar/inspecionar o APK.
- VS Code + extensão CodeQL para executar interrogações QL sobre uma base de dados CodeQL.
- Docker para correr OWASP Juice Shop.
- OWASP ZAP para proxy/intercepção e fuzzing.

2. Ataques em aplicações móveis (Lab4-ess.apk)

A aplicação pede uma frase ao utilizador e valida-a no método MainActivity.verify(View v). Pela análise estática ao bytecode (strings do DEX), é possível recuperar os artefactos criptográficos usados e decifrar a frase correta.

2.1 Artefactos encontrados (frase cifrada, chave e algoritmo)

No APK foram observadas as seguintes constantes relevantes:

Artefacto	Valor
Frase cifrada (Base64)	5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=
Chave (hex, 16 bytes)	8d127684cbc37c17616d806cf50473cc
Algoritmo / modo / padding	AES / ECB / PKCS7Padding (equivalente a PKCS5Padding em Java)

Decifrando o ciphertext Base64 com AES-ECB e a chave acima, obtém-se a frase correta:

I want to believe

2.2 Resolução com DecodeAndroidKey.java

O programa foi completado preenchendo ciphertext, chave e transformação do Cipher.

Código java modificado:

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class DecodeAndroidKey_solved {
    private static final String ENCRYPTED_BASE64 =
        "5UJiFctbmgbdOLXmpL12mkno8HT4Lv8dlat8FxR2G0c=";
    private static final String KEY_HEX =
        "8d127684cbc37c17616d806cf50473cc";

    public static void main(String[] args) throws Exception {
        byte[] encryptedBytes = Base64.getDecoder().decode(ENCRYPTED_BASE64);
        byte[] keyBytes = hexStringToByteArray(KEY_HEX);

        SecretKeySpec secretKey = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);

        byte[] decrypted = cipher.doFinal(encryptedBytes);
        System.out.println("Descodificado: " + new String(decrypted));
    }

    private static byte[] hexStringToByteArray(String hex) {
        int len = hex.length();
        byte[] out = new byte[len / 2];
        for (int i = 0; i < len; i += 2) {
            out[i / 2] = (byte) ((Character.digit(hex.charAt(i), 16) << 4)
                + Character.digit(hex.charAt(i + 1), 16));
        }
        return out;
    }
}
```

Saída ao executar:

Descodificado: I want to believe

3. CodeQL — análise local (vulnapp-ess.zip + vulnapp_db.zip)

Foi carregada a base de dados CodeQL (a partir do arquivo vulnapp_db.zip) no workspace do starter kit. Em seguida foram localizadas e executadas as queries correspondentes a CWE-79, CWE-89 e CWE-327.

CWE	Query CodeQL (ID)	O que deteta (resumo)
CWE-79	java/xss	Cross-Site Scripting (XSS) — dados não confiáveis refletidos/armazenados em HTML.
CWE-89	java/sql-injection	SQL Injection — queries construídas a partir de input controlado pelo utilizador.
CWE-327	java/weak-cryptographic-algorithm	Uso de algoritmos criptográficos fracos/quebrados (ex.: DES, MD5, SHA-1, ECB).

3.1 CWE-79 (XSS)

Local no código: `pt.isel.vulnerableapp.controller.SearchController`.

Source: input do utilizador vindo de `HttpServletRequest.getParameter("query")`.

Sink: escrita direta no HTML gerado via `PrintWriter.println` sem encoding/escaping.

Exploração: pedido HTTP do tipo `/search?query=<script>alert(1)</script>` reflete o payload na resposta em ambiente controlado.

3.2 CWE-89 (SQLi)

Local no código: `pt.isel.vulnerableapp.controller.UserController`.

Source: valores de username vindos de parâmetros HTTP (ex.: `request.getParameter("username")`).

Sink: construção de SQL por concatenação e execução por:

`Statement.executeQuery()/executeUpdate()`.

Exploração: inserir caracteres de controlo SQL no username (ex.: ' OR '1'='1) pode permitir ler dados indevidos ou contornar validações, dependendo da query.

3.3 CWE-327 (cripto fraca)

Local no código: `pt.isel.vulnerableapp.util.SecurityUtils`.

Source: dados sensíveis (p.ex. passwords) passados como `String/byte[]` para funções utilitárias.

Sink: chamadas a APIs de criptografia com algoritmos fracos, como `MessageDigest.getInstance("MD5")`, `MessageDigest.getInstance("SHA-1")`, `Cipher.getInstance("DES/ECB/PKCS5Padding")`.

Exploração: MD5/SHA-1 são vulneráveis a ataques de colisão; DES pode ser atacado com bruteforce; ECB expõe padrões.

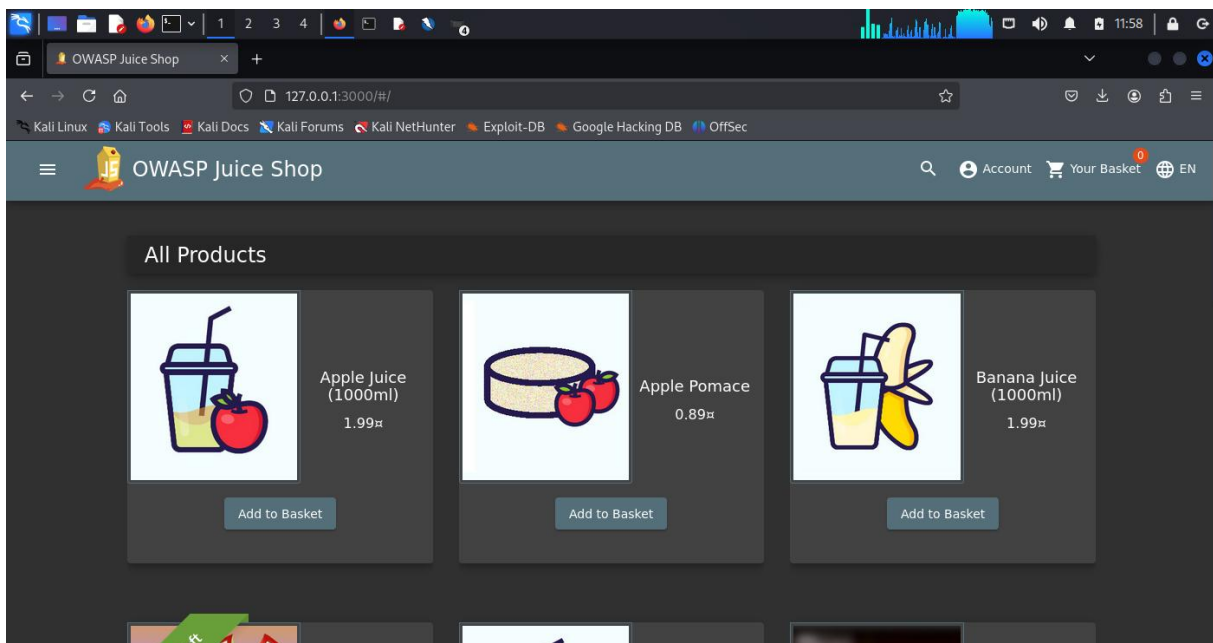
Isto reduz o custo de ataque e pode comprometer confidencialidade/integridade.

4. Análise dinâmica — OWASP Juice Shop

A aplicação foi executada localmente em Docker e analisada com DevTools e OWASP ZAP.

4.1 Execução em Docker

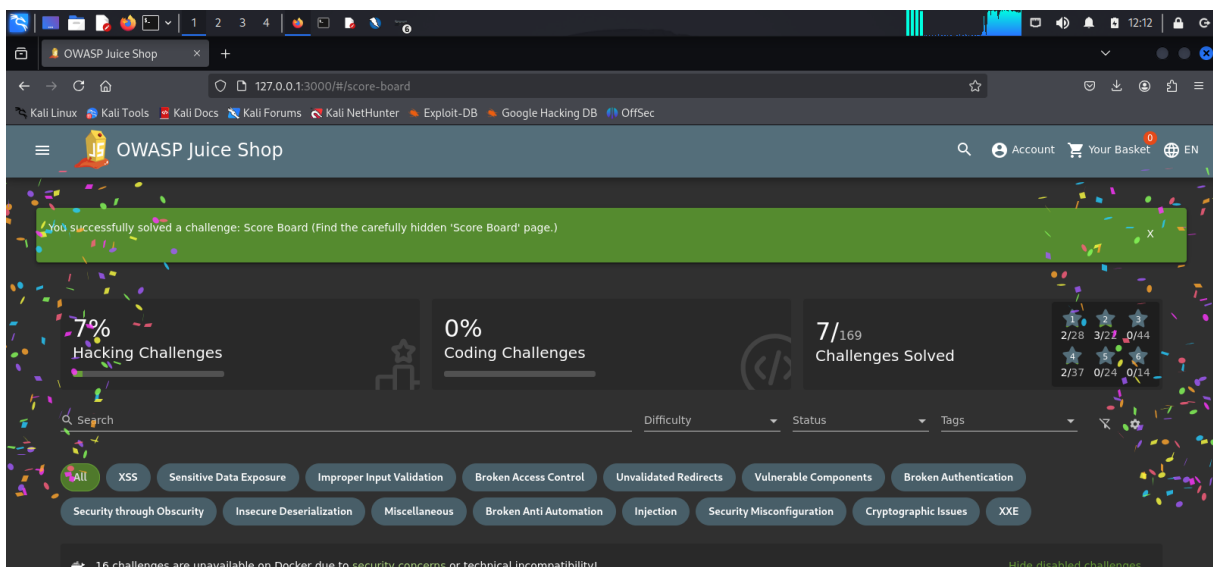
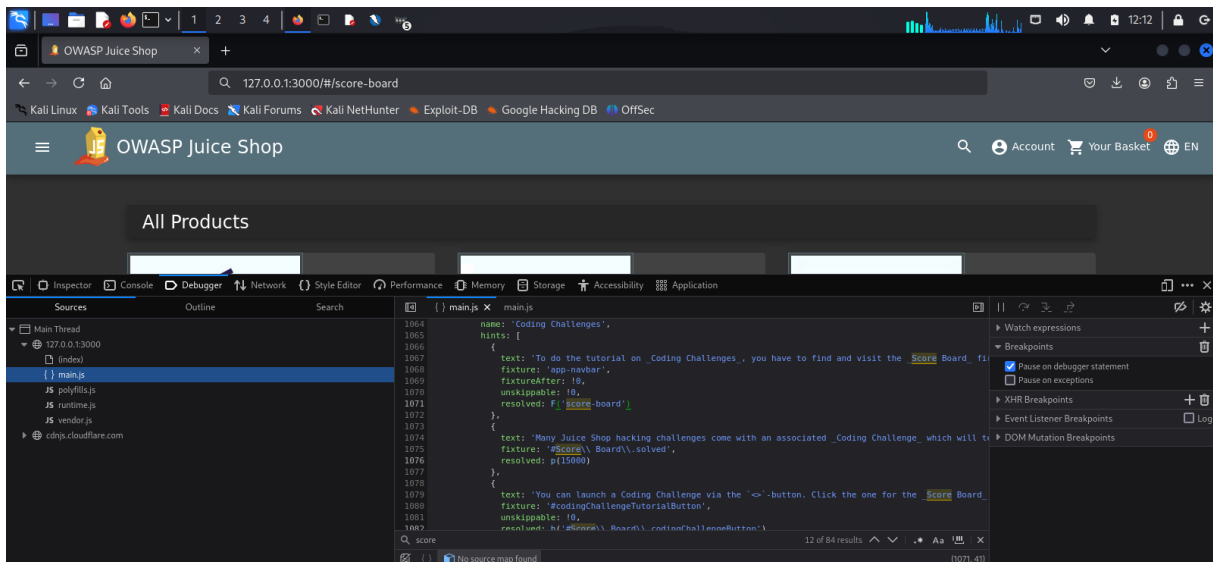
```
docker run -d -p 3000:3000 bkimminich/juice-shop
```



4.2 Descobrir o Score Board (rota escondida)

Com DevTools (F12) → Sources → abrir main.js e pesquisar por "score". A rota encontrada permite aceder a:

<http://localhost:3000/#/score-board>



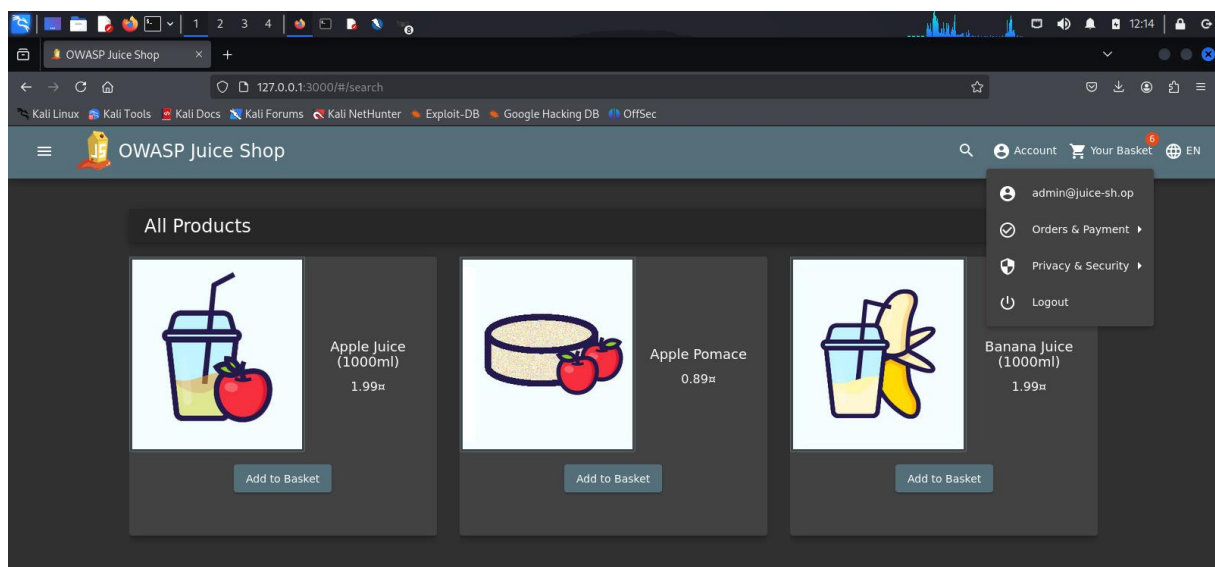
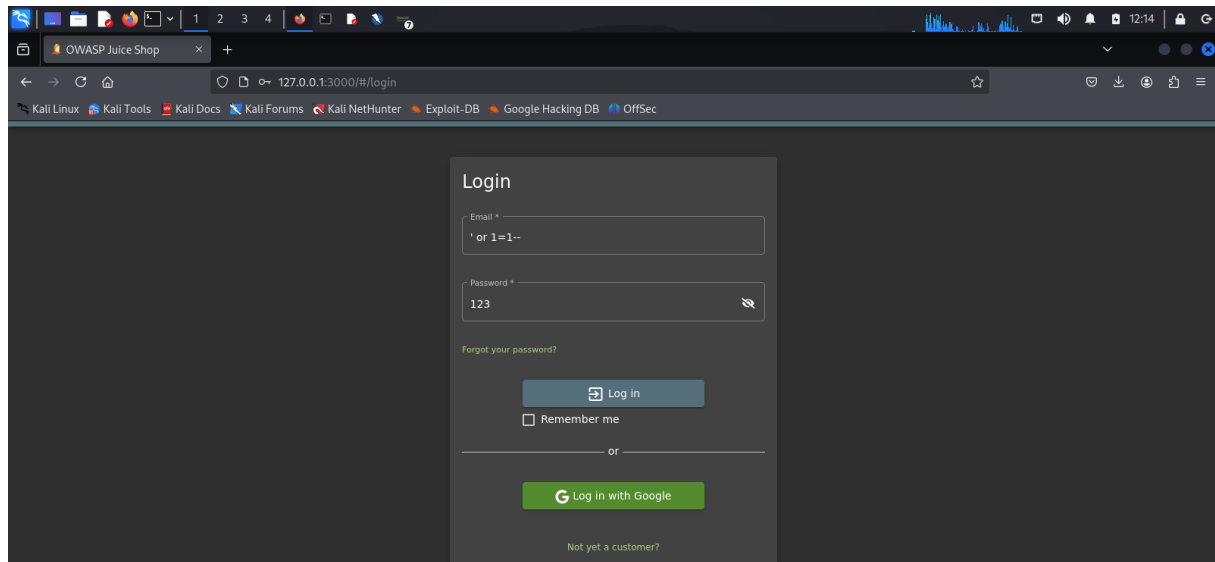
4.3 SQL Injection no login (admin)

No formulário de login, injetou-se no campo Email um payload que force a query a devolver o primeiro registo (admin) ou que comente a validação.

Email: ' or 1=1--
Password: qualquer

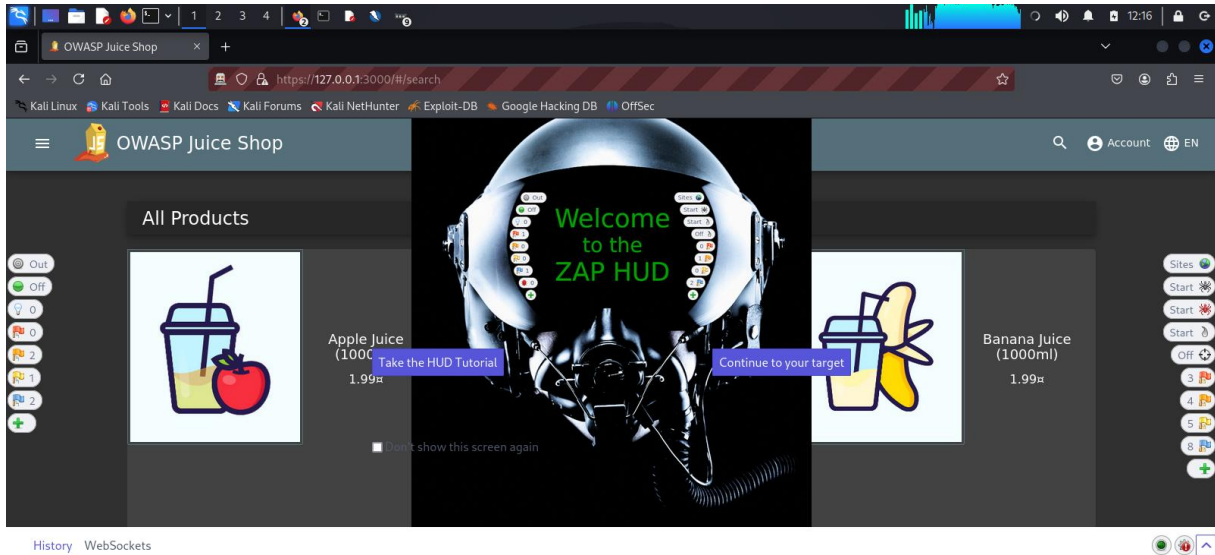
ou

Email: admin@juice-sh.op'--
Password: qualquer



4.4 ZAP — Manual Explore

Abrir ZAP → Quick Start → Manual Explore → lança browser (preferência Firefox) e navegar para o Juice Shop através desse browser, o tráfego aparece no History do ZAP.



4.5 Fuzzing da password admin###

Interceptar um pedido de login válido (POST /rest/user/login), enviou-se para o Fuzzer e aplicamos payloads no campo password com lista admin000...admin999. A credencial obtida foi:

admin@juice-sh.op / admin123

The screenshot displays the ZAP 2.17.0 interface during a fuzzing session. The main window shows the 'Fuzz Locations' tab with a POST request to `http://localhost:3000/rest/user/login`. The request body is a JSON object: `{ "email": "admin@juice-hp.op", "password": "123456" }`. The 'Add Payload' dialog is open, showing a list of files to be added to the fuzzing payload, including `john.txt`, `phpbb.txt`, `twitter.txt`, and `weaksource.txt`. The 'Payloads Preview' section shows a list of payloads, including `1: 12345`, `2: abc123`, `3: password`, `4: computer`, `5: 123456`, `6: tiger`, `7: 1234`, `8: alb2c3`, `9: qwerty`, `10: 123`, `11: xxx`, `12: money`, `13: test`, `14: carmen`, `15: mickey`, `16: secret`, `17: summer`, `18: internet`, `19: service`, `20: canada`, `21: hello`, `22: ranger`, and `23: shadow`.

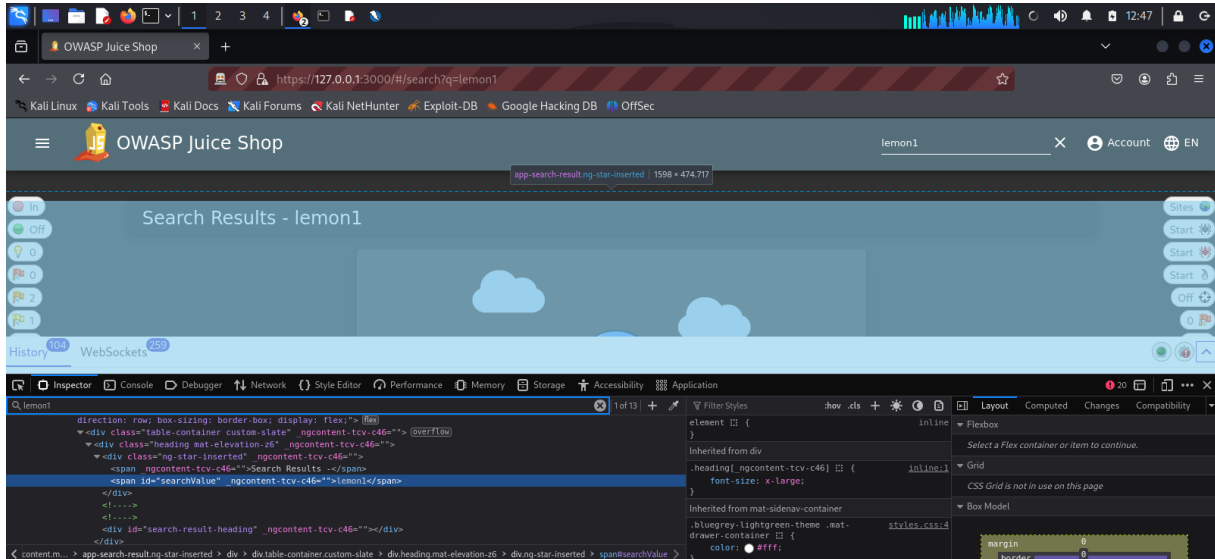
The 'Payloads' dialog is also open, showing a list of payloads with columns for '#', 'Type', 'Description', and '# of Processors'. The 'Remove without confirmation?' checkbox is checked.

The bottom panel shows the 'Alerts' tab with a table of results:

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
0	Original	200	OK	607 ms	386 bytes	799 bytes			admin123
1	Fuzzed	401	Unauthorized	25 ms	387 bytes	26 bytes			12345
2	Fuzzed	401	Unauthorized	748 ms	387 bytes	26 bytes			abc123
3	Fuzzed	401	Unauthorized	219 ms	387 bytes	26 bytes			password

4.6 Alterar termo de pesquisa na barra de endereço

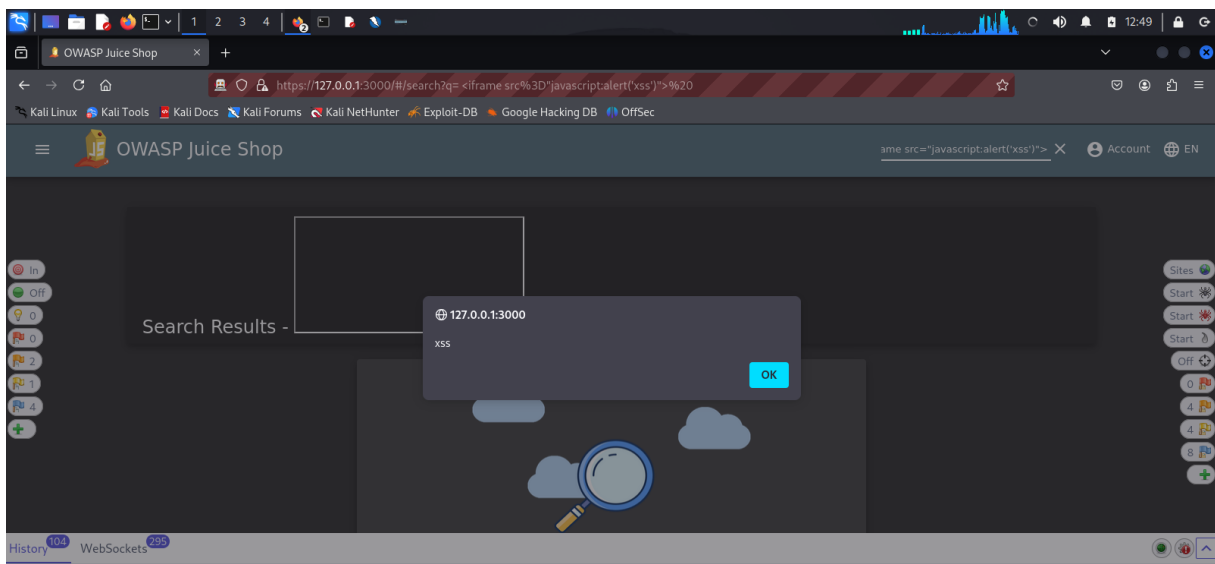
Após pesquisar por Lemon (/search?q=Lemon), alterou-se para Lemon1. O termo é apresentado num elemento `` com binding `[innerHTML]`, permitindo manipulação via DOM/URL.



4.7 DOM XSS

Inseriu-se no campo de pesquisa um payload do tipo `<iframe src="javascript:alert('xss')">`.

O XSS ocorre porque o valor de pesquisa é injetado no DOM como HTML (via `innerHTML`) sem sanitização suficiente.



4.8 Persisted XSS via API

Correu-se o container com `NODE_ENV=unsafe`.

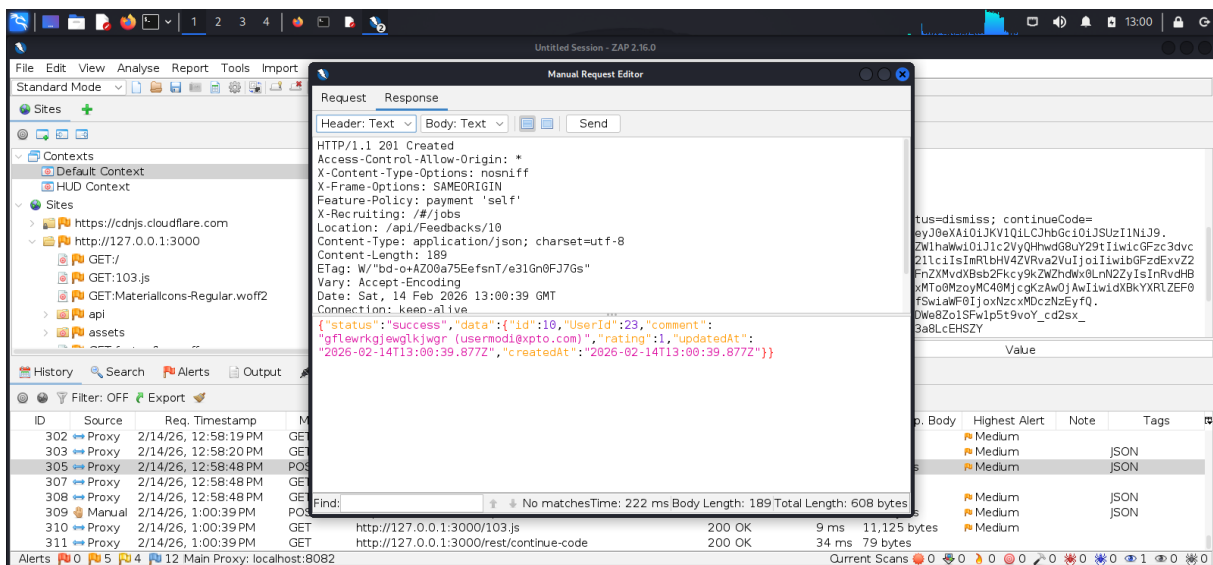
Exploração:

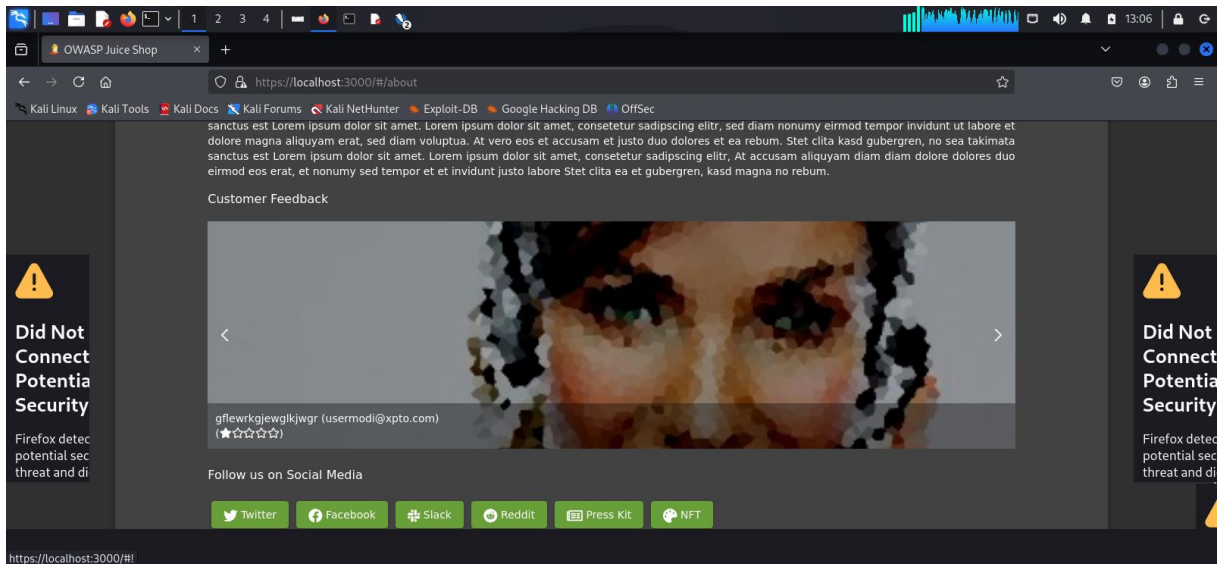
- Obteve-se um token (Authorization: Bearer)
- Enviou-se POST para `/api/Products` criando produto com description igual a um payload HTML
- Visitar `/search` para disparar o XSS.

Mesmo com HTML injection, uma CSP estrita pode bloquear execução de scripts inline/event-handlers e URLs javascript, reduzindo impacto e dificultando exfiltração.

4.9 Feedback em nome de outro utilizador

Usando o ZAP como proxy, submeteu-se feedback normal no Contact Us, interceptou-se o pedido e modificou-se no requester o json o ID de outro utilizador antes de reenviar.





4.10 Roubo do cookie/token

Um atacante pode convencer a vítima a abrir um link/página dentro do domínio vulnerável que contenha um payload XSS.

Quando o script executa, ele pode ler o token de sessão caso esteja acessível via JavaScript, através da cookie sem HttpOnly ou token em localStorage, e, enviá-lo para um servidor controlado pelo atacante, exfiltração.

Com o token, o atacante pode autenticar-se como a vítima e invocar endpoints protegidos.

As mitigações incluem HttpOnly, SameSite, CSP, sanitização/encoding e validação no servidor.

5. Conclusão

O laboratório permitiu exercitar a análise estática e dinâmica em aplicações web como a detecção de XSS/SQLi/cripto fraca com CodeQL e exploração controlada com DevTools/ZAP.

As atividades reforçam a importância de validação e sanitização de entrada de dados, uso de queries parametrizadas e adoção de criptografia moderna.

Referências

- Enunciado do Laboratório 4 (ESS).
- OWASP Juice Shop — Appendix / Solutions (Score Board, SQLi login, XSS, API-only XSS, feedback spoofing).
- OWASP Juice Shop — Companion Guide (unsafe mode / NODE_ENV=unsafe).
- MDN Web Docs — Content-Security-Policy (CSP) e diretiva script-src.
- CodeQL Query Help (java/xss, java/sql-injection, java/weak-cryptographic-algorithm).