

Projeto 1 SIO

Vulnerabilidades em produtos de software

Bernardo Pinto – 105926

Diogo Falcão – 108712

João Santos – 110555

Matilde Teixeira – 108193

(Turma P5)

Índice

Introdução	3
Funcionalidades da loja online	4
O que é uma Common Weakness Enumeration (CWE) ?	4
Common Weakness Enumerations (CWEs)	5
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	5
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	7
CWE-522: Insufficiently Protected Credentials	9
CWE-521: Weak Password Requirements	12
CWE-284: Improper Access Control	13
CWE-434: Unrestricted Upload of File with Dangerous Type	15
CWE-20: Improper Input Validation	17
CWE-839: Numeric Range Comparison without Minimum Check	18
CWE-209: Generation of Error Message Containing Sensitive Information	19
CWE-1284: Improper Validation of Specified Quantity in Input	20
Conclusão	22
Referências	23

Introdução

Este documento descreve a implementação do "Projeto 1", que se concentra nas vulnerabilidades em produtos de software. O primeiro passo deste projeto envolveu o desenvolvimento de uma loja online especializada em produtos do DETI, que incluem canecas, copos, t-shirts, hoodies e camisolas.

O objetivo principal foi criar uma loja funcional que consiste em duas versões: a primeira versão - "app" - que apresenta um conjunto de 10 vulnerabilidades, e a segunda versão - "app_sec" - que representa a variante segura da loja.

Ao longo deste documento, serão apresentadas todas as Common Weakness Enumerations (CWEs) incorporadas neste projeto, bem como as suas descrições e correções, além de uma breve introdução das funcionalidades da aplicação.

Funcionalidades da loja online

A nossa loja online permite a compra de produtos, seja adicionando-os ao carrinho ou realizando a compra imediatamente. Além disso, no carrinho, é assegurada a modificação da quantidade de produtos (adicionando ou removendo produtos). A loja oferece também a funcionalidade de criação de contas e a realização do login como administrador ou cliente, consoante o tipo de conta.

Desta forma a aplicação faculta a execução de comentários na página de cada produto, com imagens e texto ou apenas texto ou apenas imagens. O site também proporciona recursos de pesquisa de produtos, não apenas por meio da barra de pesquisa, mas também por meio de filtros disponibilizados em Category.

A base utilizada para o desenvolvimento das vulnerabilidades deste site foi um repositório de outro autor, Shashi Rash, que utilizou diversas tecnologias:

- No Front-End foram utilizados HTML, CSS, JavaScript e Bootstrap;
- No Back-End, foram utilizados as seguintes tecnologias: Java(JDK8+), JDBC (Java Database Connectivity, que facilita comunicação com a base de dados por uma API), Servlet e JSP (Jakarta Server Pages).
- Para a base de dados foi utilizado MySQL. Para além disto foi utilizado TomCat, para as Java Server Pages (JSP) e Servlet, Maven, docker-compose, para facilitar a execução do carregamento da página.

O que é uma Common Weakness Enumeration (CWE) ?

Uma lista de Common Weakness Enumeration, em português, Enumeração Comum de Vulnerabilidades, são vários tipos de falhas e vulnerabilidades de segurança de software. O principal objetivo desta lista é descrever e categorizar as diferentes fraquezas e vulnerabilidades encontradas nos sistemas. Cada elemento da lista é composto por um identificador numérico, por um título, por uma descrição detalhada da fraqueza e também com exemplos de como esta falha pode ser explorada e mitigada. Desenvolvidas em 2006 pelo *MITRE* (Massachusetts Institute of Technology Research and Engineering), as CWEs encontram-se disponíveis no site. A transparência ajuda assim a comunidade a colaborar e a tomar medidas para mitigar esses problemas.

Common Weakness Enumerations (CWEs)

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Esta CWE é denominada pela mitre.org como uma das CWEs mais insistentes nos últimos cinco anos. Isto porque é uma CWE que está representada sempre no topo das vulnerabilidades ano após ano, apesar do esforço que se tem feito para reduzir o número de ataques causados por esta na comunidade.

A CWE-89, também conhecida por “SQL INJECTION” foi classificada como a 3ª mais usada em ataques em 2023. Acontece quando não existe validação de todos os inputs feitos pelo lado do cliente e por isso, o atacante usa elementos especiais capazes de modificar o comando SQL inicial pretendido para realizar operações sobre a base de dados.

Esta falta de neutralização do input do utilizador pode permitir modificar, eliminar, alterar e contornar verificações de segurança na base de dados. Esta vulnerabilidade, mais especificamente do tipo “login bypass attack” encontra-se no ficheiro *UserServiceImpl.java*, na função *isValidCredential()*.

```
103  @Override
104  public String isValidCredential(String emailId, String password) {
105      //String status = "Login Denied! Incorrect Username or Password";
106      String status = "Please input the correct login data";
107
108      Connection con = DBUtil.provideConnection();
109
110      PreparedStatement ps = null;
111      ResultSet rs = null;
112
113      try {
114
115          String sql = "SELECT * FROM user WHERE email='" + emailId + "' and password='" + password + "'";
116
117          ps = con.prepareStatement(sql);
118
119          rs = ps.executeQuery();
120
121          if (rs.next())
122              status = "valid";
123
124      } catch (SQLException e) {
125          status = "Error: " + e.getMessage();
126          e.printStackTrace();
127      }
128
129      DBUtil.closeConnection(con);
130      DBUtil.closeConnection(ps);
131      DBUtil.closeConnection(rs);
132      return status;
133  }
```

Figura 1

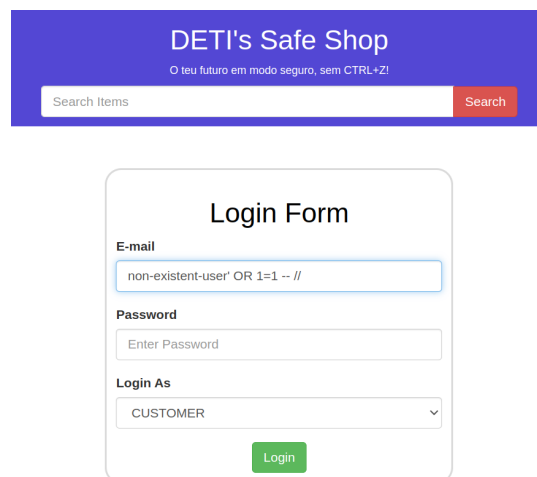
Como é possível observar, a query a ser executada por parte do server na app é não parametrizada e resulta de uma concatenação de strings e variáveis. Desta forma, a query de SQL apenas funciona se o *emailId* e a *password* não contenham nenhum caractere do tipo ‘ (apóstrofe).

Se por exemplo um atacante usar o seguinte input no campo “e-mail” do login form:

non-existent-user' OR 1=1 -- //

então a query passar a ter o formato:

"SELECT * FROM user WHERE email=" + non-existent-user' OR 1=1 -- // + " and password=" + password + "".



The image shows a screenshot of a web application interface. At the top, there is a purple header with the text "DETI's Safe Shop" and a subtitle "O teu futuro em modo seguro, sem CTRL+Z!". Below the header is a search bar with the placeholder text "Search Items" and a red "Search" button. Below the search bar is a "Login Form" with the following fields: "E-mail" (containing the text "non-existent-user' OR 1=1 -- //"), "Password" (containing the placeholder text "Enter Password"), and "Login As" (a dropdown menu with "CUSTOMER" selected). A green "Login" button is at the bottom of the form.

Figura 2

Com a adição de *non-existent-user' OR 1=1 -- //*, faz-se com que a query a condição SQL seja inquestionavelmente verdadeira, uma vez que esta passa a pedir para que se selecione da tabela user quando o email ser “non-existent” ou 1=1, ou seja, verdadeiro e os dois hifens comentam o resto da query para a frente. Isto permite ao hacker entrar na app sem ter nenhum registo prévio.

Para corrigir esta vulnerabilidade, usámos queries parametrizadas onde em vez de fazermos a concatenação destas, colocámos pontos de interrogação, correspondentes aos nomes das variáveis declaradas a seguir:

```

113     try {
114         String sql = "SELECT * FROM user WHERE email = ?";
115         ps = con.prepareStatement(sql);
116         ps.setString(1, emailId);
117
118         rs = ps.executeQuery();
119
120         if (rs.next()) {
121             String hashedPasswordFromDB = rs.getString("password_hash"); // Recupere o hash da senha do banco de dados
122             if (BCrypt.checkpw(password, hashedPasswordFromDB)) {
123                 status = "valid"; // Senha está correta
124             } else {
125                 status = "Login Denied! Incorrect Username or Password"; // Senha incorreta
126             }
127         } else {
128             status = "Login Denied! Incorrect Username or Password"; // Usuário não encontrado
129         }

```

Figura 3

Nota: na figura 3, como já referido, é possível observar a parametrização da query. Aponta-se apenas que a 2ª variável (password) que permite acesso à app, passa por outro processo de parametrização e decodificação, correspondente a outro CWE que iremos mencionar mais à frente. Em função disto, a versão segura da app retorna um erro e não incorre nos mesmos erros, como é possível verificar na figura 4.

The image shows two parts of a web application. The top part is a header for 'DETI's Safe Shop' with a tagline 'O teu futuro em modo seguro, sem CTRL+Z!' and a search bar. The bottom part is a 'Login Form' with a green error message: 'Login Denied! Incorrect Username or Password'. The form includes fields for 'E-mail', 'Password', and a 'Login As' dropdown menu set to 'CUSTOMER', with a green 'Login' button at the bottom.

Figura 4

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

A CWE-79 - Improper Neutralization of Input During Web Page Generation, mais conhecida como Cross-site Scripting, refere-se à falha do tratamento dos dados inseridos pelo utilizador e da inserção direta na página web sem realizar nenhuma validação. A exploração mais comum desta falha em páginas web é feita através da escrita de scripts nos campos de preenchimento.

Esta falha, no nosso website, pode ser explorada na seção de comentários de um produto, o atacante comenta um script, neste caso “<script>alert(“XSS”)</script>”, e a página inclui diretamente no html, interpretando o mesmo como código html e correndo o script.

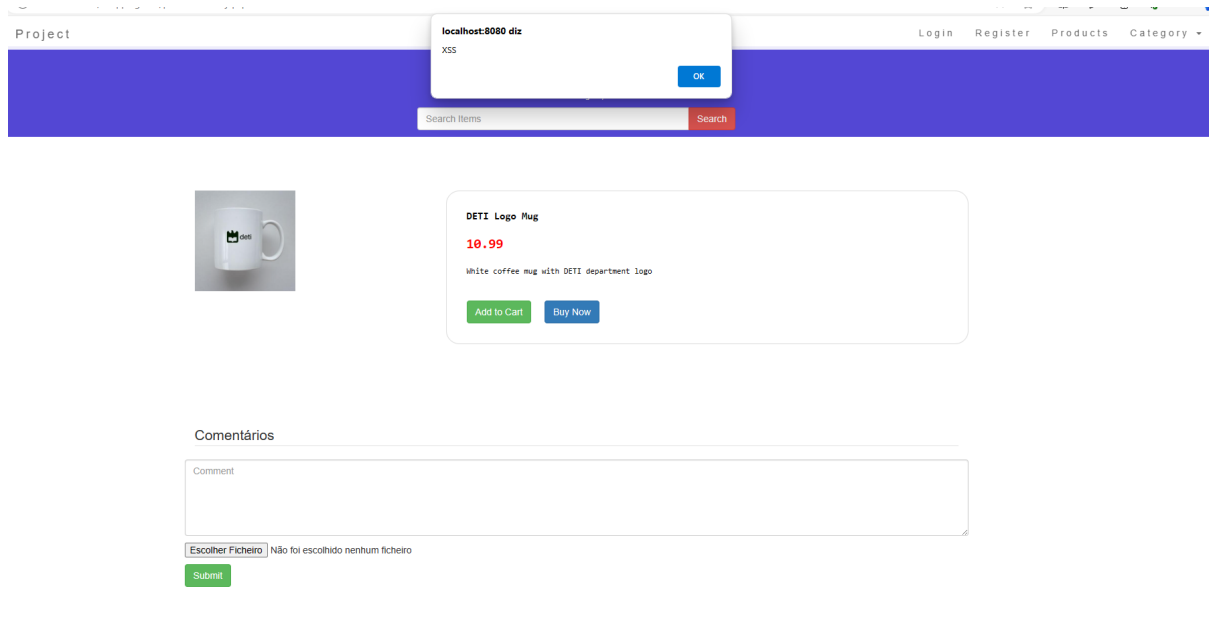


Figura 5

Neste caso, para resolver este problema, modificamos a função responsável por mostrar os comentários. Inicialmente a função acrescenta diretamente o input do utilizador ao elemento html.

```
commentDiv.html(entry.comment);
```

Figura 6

Para resolver isso, na versão segura adicionamos o input do utilizador através do método “`.textContent`”, pois este método interpreta o input como texto puro e não como elemento html.

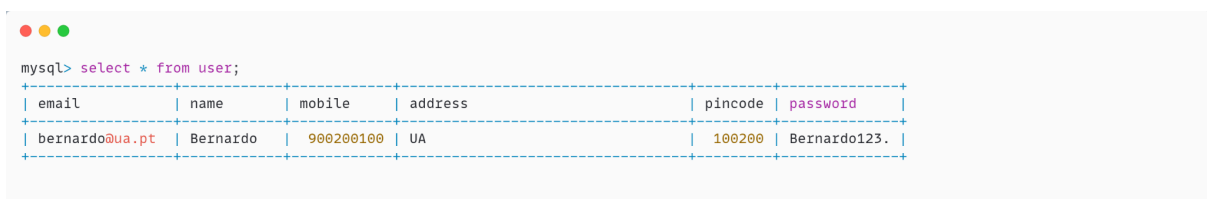
```
commentDiv.textContent = entry.comment;
```

Figura 7

CWE-522: Insufficiently Protected Credentials

A vulnerabilidade identificada, classificada como CWE-522, diz respeito a situações em que informações de autenticação, como senhas, tokens ou chaves, não recebem a devida proteção, o que pode resultar em sérias vulnerabilidades de segurança e potenciais violações de dados.

No contexto do nosso sistema, a vulnerabilidade estava relacionada à tabela *User* da base de dados e à lógica de armazenamento utilizada. Quando um novo utilizador se regista na plataforma, a senha era armazenada na base de dados sem qualquer medida de segurança, como evidenciado na consulta a seguir:



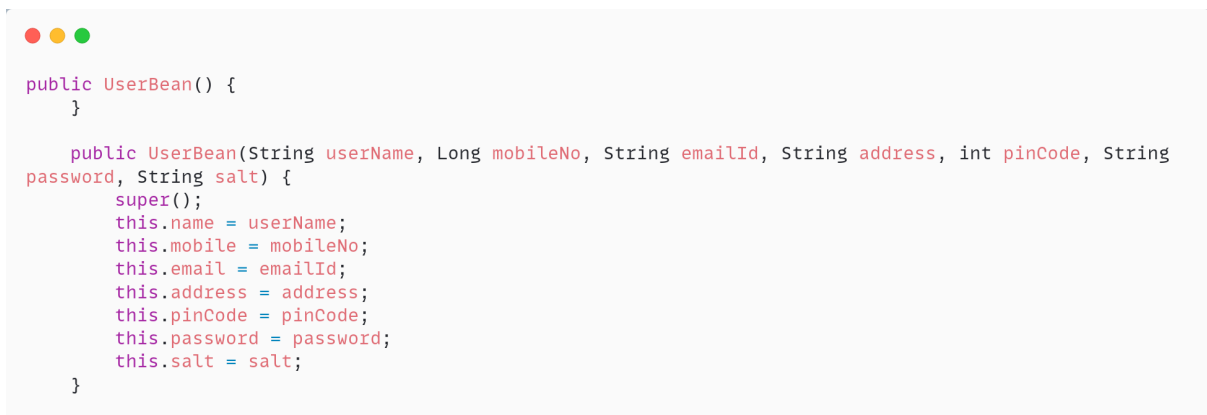
```
mysql> select * from user;
```

email	name	mobile	address	pincode	password
bernardo@ua.pt	Bernardo	900200100	UA	100200	Bernardo123.

Figura 8

Por conseguinte, para resolver esta vulnerabilidade, foi utilizada a biblioteca "*jBCrypt*", que desempenha um papel crucial ao lidar com a segurança das senhas, fazendo uso do algoritmo de hash de senhas *BCrypt*. Essa abordagem foi adotada para garantir o armazenamento seguro das senhas, uma vez que o BCrypt é projetado especificamente para esta finalidade, incorporando práticas como o salting e a capacidade de ajustar o custo do processo de hash, tornando-o mais resistente a ataques forçados.

Para implementar a lógica da biblioteca, foi essencial a inclusão do atributo *Salt*, que é usado para aumentar a proteção das senhas dos utilizadores, no *UserBean*. Assim, mesmo que duas pessoas tenham a mesma senha, os seus hashes serão distintos devido à inclusão desse valor único.



```
public UserBean() {  
    }  
  
    public UserBean(String userName, Long mobileNo, String emailId, String address, int pinCode, String password, String salt) {  
        super();  
        this.name = userName;  
        this.mobile = mobileNo;  
        this.email = emailId;  
        this.address = address;  
        this.pinCode = pinCode;  
        this.password = password;  
        this.salt = salt;  
    }  
}
```

Figura 9

Com isso, também foi necessário alterar o *UserServiceImpl*, uma vez que o campo *Salt* estará presente também na base de dados, e para que não fosse necessário alterar a lógica já utilizada.

```
public String registerUser(UserBean user) {

    String status = "User Registration Failed!";

    boolean isRegtd = isRegistered(user.getEmail());

    if (isRegtd) {
        status = "Email Id Already Registered!";
        return status;
    }
    Connection conn = DBUtil.provideConnection();
    PreparedStatement ps = null;
    if (conn != null) {
        System.out.println("Connected Successfully!");
    }

    try {

        ps = conn.prepareStatement("insert into " + IUserConstants.TABLE_USER + " values(?,?,?,?,?,?,?)");

        ps.setString(1, user.getEmail());
        ps.setString(2, user.getName());
        ps.setLong(3, user.getMobile());
        ps.setString(4, user.getAddress());
        ps.setInt(5, user.getPinCode());
        ps.setString(6, user.getPassword());
        ps.setString(7, user.getSalt());

        int k = ps.executeUpdate();

        if (k > 0) {
            status = "User Registered Successfully!";
            // MailMessage.registrationSuccess(user.getEmail(), user.getName().split(" ")[0]);
        }

    } catch (SQLException e) {
        status = "Error: " + e.getMessage();
        e.printStackTrace();
    }

    DBUtil.closeConnection(ps);
    DBUtil.closeConnection(ps);

    return status;
}
```

Figura 10

Por fim, a biblioteca de criptografia foi incorporada no *RegisterSrv*, eliminando a vulnerabilidade, uma vez que as senhas originais não são mais acessíveis em possíveis ataques.

```
import org.mindrot.jbcrypt.BCrypt;

@WebServlet("/RegisterSrv")
public class RegisterSrv extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        String userName = request.getParameter("username");
        Long mobileNo = Long.parseLong(request.getParameter("mobile"));
        String emailId = request.getParameter("email");
        String address = request.getParameter("address");
        int pinCode = Integer.parseInt(request.getParameter("pincode"));
        String password = request.getParameter("password");
        String confirmPassword = request.getParameter("confirmPassword");
        String status = "";
        if (password != null && password.equals(confirmPassword)) {

            String salt = BCrypt.gensalt();

            String hashedPassword = BCrypt.hashpw(password, salt);

            UserBean user = new UserBean(userName, mobileNo, emailId, address, pinCode, hashedPassword, salt);

            UserServiceImpl dao = new UserServiceImpl();

            status = dao.registerUser(user);
        } else {
            status = "Password not matching!";
        }

        RequestDispatcher rd = request.getRequestDispatcher("register.jsp?message=" + status);
        rd.forward(request, response);
    }
}
```

Figura 11

Portanto, a partir do momento que a vulnerabilidade foi resolvida, os dados estão acessíveis da seguinte forma:

```
mysql> select * from user;
```

email	name	mobile	address	pincode	password_hash	salt
bernardo@ua.pt	Bernardo	900200100	UA	100200	\$2a\$10\$uwB1qEenpNIZyxd5jpMjD.k8ZAnfoVn2U	\$2a\$10\$uwB1qEenp

Figura 12

CWE-521: Weak Password Requirements

A CWE 521-Weak Password Requirements, como o nome indica, diz respeito a passwords, e ao grau de segurança associado a estas. Ao utilizar passwords seguras, previne-se ataques que possam comprometer as contas dos utilizadores. Esta vulnerabilidade torna possível a utilização por parte dos utilizadores, palavras-passes mais fracas, mais suscetíveis a ataques por parte de terceiros.

Na versão insegura do nosso site, isto verifica-se no sign up da página de cliente, onde o utilizador, ao fazer a sua conta, não tenha quaisquer parâmetros para se guiar e a tornar segura, tornando-a propícia a ataques, (por exemplo: o utilizador podia introduzir uma palavra passe de um só carácter). De modo a corrigir esta vulnerabilidade, introduzimos os seguintes parâmetros de verificação, através do método *pattern*. A palavra-passe tem de conter:

- Pelo menos um número;
- Uma letra maiúscula e minúscula;
- Conter no mínimo 8 caracteres.

```
<input type="password" name="password" class="form-control" id="password"
      name="last_name" pattern="(?!.*\d)(?!.*[a-z])(?!.*[A-Z]).{8,}"
      title="Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters" required>
```

Figura 13

Sendo assim, quando o utilizador tentar colocar uma password que não corresponda a estes critérios na versão da app segura, exibe-se o aviso referente ao(s) critério(s) em falta.

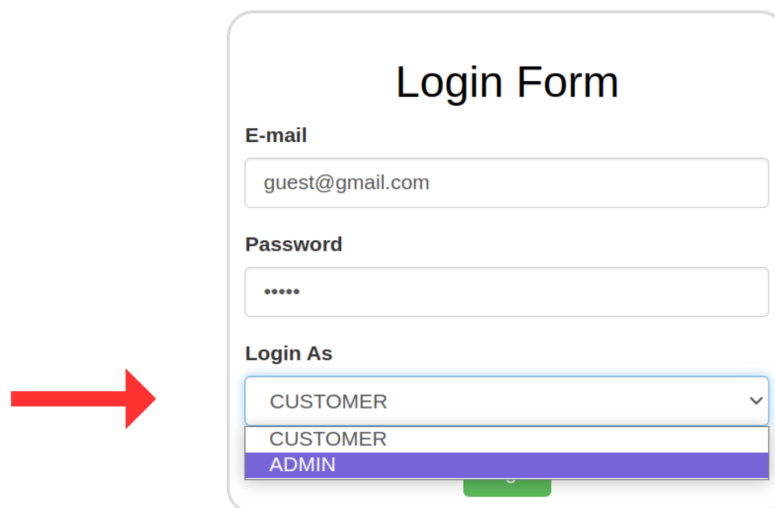
The screenshot shows the 'Registration Form' for 'DETI's Safe Shop'. The form includes fields for Name, Email, Address, Mobile, Pin Code, Password, and Confirm Password. The Password field is highlighted with a red border, and a red error message box is displayed below it. The error message states: 'É preciso que o formato corresponda ao exigido. Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters'. The form also features a search bar at the top and a green 'Register' button at the bottom right.

Figura 14

CWE-284: Improper Access Control

A CWE-284 (Improper Access Control) é uma vulnerabilidade que ocorre quando um sistema não restringe ou limita o acesso de recursos da aplicação a terceiros. Esta vulnerabilidade de controlo sucede-se devido à insuficiência ou falta de autenticação, autorização e/ou responsabilidade. Figura 13

Um exemplo é a falta de restrições específicas em certas funções de uma aplicação, como a possibilidade de aceder ao papel de administrador da aplicação quando não devia ser possível (como é o caso na nossa versão insegura da aplicação). Isso pode permitir que pessoas sem as devidas permissões possam modificar, excluir ou alterar a quantidade de produtos, ou até mesmo modificar pedidos na nossa aplicação. Na página do login, ao seleccionar **ADMIN**, como o **role** com que se quer entrar no sistema, a aplicação não verifica se o e-mail e a password introduzidas fazem parte da relação de e-mails e passwords de administradores. Logo, qualquer utilizador poderá passar a ter o **role** de administrador, desde que tenha conta na aplicação.



The diagram shows a 'Login Form' with three input fields: 'E-mail' (containing 'guest@gmail.com'), 'Password' (masked with dots), and 'Login As' (a dropdown menu). A red arrow points to the 'Login As' dropdown. The dropdown menu is open, showing two options: 'CUSTOMER' and 'ADMIN'. The 'ADMIN' option is highlighted in blue, indicating it is the selected role.

Figura 15

Devem-se gerir as definições da aplicação minuciosamente e definir privilégios distintos para cada **role** de utilizador. Dados sensíveis não devem ser permitidos ser vistos por utilizadores comuns, muito menos gerenciados. Isto pode levar a divulgação de dados privados, uso de funções exclusivas a administradores e apropriação de contas de utilizadores. Com isto, adicionamos as respetivas verificações no servlet que lida com o processo de login (**LoginSrv** em **LoginSrv.java**):

```

41     if (userType.equals("admin")) { // Login as Admin
42
43         status = udao.isValidCredential(userName, password);
44
45         if (status.equalsIgnoreCase("valid")) {
46             // valid
47
48             RequestDispatcher rd = request.getRequestDispatcher("adminViewProduct.jsp");
49
50             HttpSession session = request.getSession();
51
52             session.setAttribute("username", userName);
53             session.setAttribute("password", password);
54             session.setAttribute("usertype", userType);
55
56             rd.forward(request, response);
57
58         } else {
59             // Invalid;
60             RequestDispatcher rd = request.getRequestDispatcher("login.jsp?message=" + statusAdmin);
61             rd.include(request, response);
62         }
63     } else { // Login as customer
64
65         if (status.equalsIgnoreCase("valid")) {
66             // valid user
67
68             UserBean user = udao.getUserDetails(userName, password);
69
70             HttpSession session = request.getSession();
71
72             session.setAttribute("userdata", user);
73
74             session.setAttribute("username", userName);
75             session.setAttribute("password", password);
76             session.setAttribute("usertype", userType);
77
78             RequestDispatcher rd = request.getRequestDispatcher("userHome.jsp");
79
80             rd.forward(request, response);
81
82         }

```

Figura 16

Na figura 16 verifica-se se o utilizador é um *administrador* ou um *cliente* com base no tipo de utilizador que está a tentar efetuar o login. Se for um *administrador* e as credenciais estiverem corretas, terá acesso aos produtos administrativos. Se for um *cliente* e as credenciais estiverem corretas, terá acesso à página inicial do cliente. Se as credenciais estiverem incorretas em qualquer caso, visualiza-se uma mensagem de erro na página de login. Para além disto, em todos os ficheiros *.jsp*, adiciona-se uma nova verificação demonstrada na Figura 17, abaixo. Esta verificação redireciona para a página de login com mensagens de erro se o tipo de utilizador não for *admin* ou se o nome de utilizador ou a palavra-passe forem nulos.

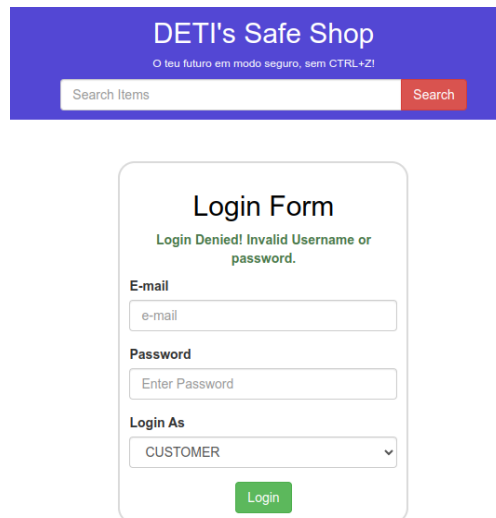
```

26     if (userType == null || !userType.equals("admin")) {
27
28         response.sendRedirect("login.jsp?message=Access Denied, Login as admin!!");
29
30     }
31
32     else if (userName == null || password == null) {
33
34         response.sendRedirect("login.jsp?message=Session Expired, Login Again!!");
35
36     }

```

Figura 17

Com efeito, a página de login, quando um utilizador comum tentar aceder ao *role* de administrador, ficará no estado possível verificar na figura 18, abaixo.



The image shows a web application header for 'DETI's Safe Shop' with a search bar. Below it is a 'Login Form' with a green error message: 'Login Denied! Invalid Username or password.' The form contains fields for 'E-mail' (with placeholder 'e-mail'), 'Password' (with placeholder 'Enter Password'), and a 'Login As' dropdown menu currently set to 'CUSTOMER'. A green 'Login' button is at the bottom.

Figura 18

CWE-434:Unrestricted Upload of File with Dangerous Type

A CWE-434, intitulada como “Unrestricted Upload File with Dangerous Type”, permite que o atacante faça upload de qualquer tipo de ficheiros pela aplicação. Estes ficheiros podem ser processados no sistema, arriscando que este ou a base de dados sofra com uma sobrecarga de ficheiros ou até conseguindo controlo do sistema através destes. Esta deve-se a uma verificação insuficiente ou inexistente do tipo de ficheiros enviados.

Na nossa aplicação, quando um utilizador quer comentar sobre um produto, este pode submeter um ficheiro (que deve ser uma foto do produto). No entanto este pode ser de qualquer formato, logo cria-se uma vulnerabilidade do sistema, ao permitir a entrada de ficheiros que não sejam imagens. Alguns exemplos são malware, ransomware, ficheiros de phishing, etc...

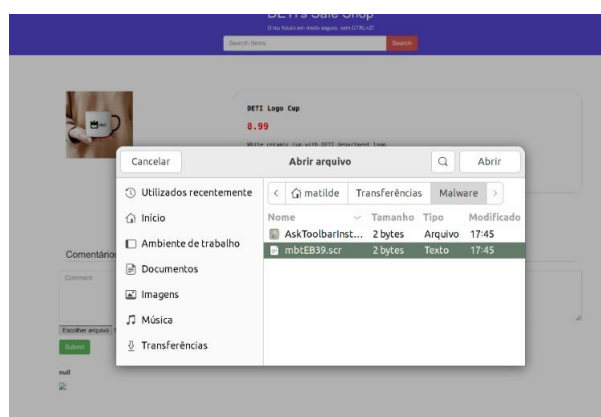


Figura 19

Para além disto, esta inserção de comentário poderia ser sem o login feito, o que trazia uma vulnerabilidade grave de segurança, em que ao invés de apresentar o nome do utilizador, o apresentava como null.



Figura 20

De modo a corrigir essa vulnerabilidade, limitamos o tipo de ficheiros que podem ser submetidos, (jpg, jpeg e png), de modo a impedir o envio de ficheiros perigosos.

```
<input type="file" alt="Submit" id="fileinsertion" accept=".jpg, .jpeg, .png">
```

Figura 21

Por forma a resolver a parte de um utilizador conseguir comentar um produto, mesmo não tendo um conta registada no site e tendo feito o login nesta, fizemos com que o comentário fosse read-only, e cujo placeholder teria informação para fazer o login, e sendo assim só quando o utilizador fizer o login é que vai conseguir comentar sobre este.

```
<textarea class="form-control" id="comment" name="comment" rows="5"
  <% if (userName == null) { %>
    readonly
    placeholder="Faça login para comentar"
  <% }
  else{%>
    placeholder="Comment"
  <%}%>
></textarea>
```

Figura 22 a)

Comentários

Faça login para comentar

Escolher arquivo Nenhum arquivo escolhido

Submit

Figura 22 b)

CWE-20: Improper Input Validation

A CWE-20: Improper Input Validation, reside sobre a validação dos dados fornecidos ao longo de todos os formulários presentes no sistema. Esta vulnerabilidade pode ser explorada de várias formas, como por exemplo na falta de: verificação de preenchimento de campos obrigatórios, verificação do limite de preenchimento dos campos ou até mesmo validação do tipo de dados fornecidos.

No caso do nosso website vulnerável, é possível realizar o registo usando dados mal formatados e logicamente incorretos como uma morada de 3 dígitos, um email que não tenha formato de email e um número de telemóvel que não contenha 9 dígitos. Isto resultará na existência de potenciais contas inoperáveis e numa acumulação de dados errados no sistema, para além de poder ter consequências negativas para os clientes também.

Para corrigir esta falha, fizemos a validação conforme o tipo de dados esperado em cada respectivo campo.

```

<div class="col-md-6 form-group">
  <label for="first_name">Name</label> <input type="text"
    name="username" class="form-control" id="first_name"
    name="first_name" required>
</div>
<div class="col-md-6 form-group">
  <label for="last_name">Email</label> <input type="email"
    name="email" class="form-control" id="last_name" name="last_name"
    required>
</div>
</div>
<div class="form-group">
  <label for="last_name">Address</label>
  <textarea name="address" class="form-control" id="last_name"
    name="last_name" required></textarea>
</div>
<div class="row">
  <div class="col-md-6 form-group">
    <label for="last_name">Mobile</label> <input type="number" maxlength="9" minlength="9"
      name="mobile" class="form-control" id="last_name"
      name="last_name" required>
  </div>
  <div class="col-md-6 form-group">
    <label for="last_name">Pin Code</label> <input type="number" maxlength="6" minlength="6"
      name="pincode" class="form-control" id="last_name"
      name="last_name" required>
  </div>
</div>

```

Figura 23

CWE-839: Numeric Range Comparison without Minimum Check

A CWE-839, intitulada “Numeric Range Comparison Without Minimum Check”, refere-se a uma categoria de vulnerabilidade de segurança de software que ocorre quando o sistema não verifica corretamente os limites numéricos de uma operação. Isto pode resultar em comportamentos inesperados do sistema, pois este pode operar sob valores que estão além dos limites seguros.

Um bom exemplo de demonstração desta falha, seria um cliente mal intencionado enviar um pedido de compra de uma quantidade em demasia de um produto específico (por exemplo 100.000 unidades). A aplicação falha ao verificar a quantidade solicitada e o sistema aceita o pedido, mesmo que o stock seja bastante menor que a quantidade especificada. Isto pode levar a uma exibição negativa de stock e incapacidade de atender outros pedidos.


Picture	Products	Price	Quantity	Add	Remove	Amount
	DETI Logo Mug	10.99	<input type="text" value="100000"/> <input type="button" value="Update"/>	<input data-bbox="938 846 962 875" type="button" value="+"/>	<input data-bbox="1145 846 1169 875" type="button" value="-"/>	1.099E7

Figura 24

Para corrigir esta insegurança é preciso garantir que a informação está coerente com a informação da base de dados.

```
ProductBean product = productDao.getProductDetails(prodId);

int availableQty = product.getProdQuantity();

PrintWriter pw = response.getWriter();

response.setContentType(type: "text/html");

if (availableQty < pQty) {

    String status = cart.updateProductToCart(userId, prodId, availableQty);

    status = "Only " + availableQty + " no of " + product.getProdName()
        + " are available in the shop! So we are adding only " + availableQty + " products into Your Cart"
        + "";

}
```

Figura 25

Guardamos na variável *availableQty* a quantidade definida na base de dados e depois executamos uma comparação simples com o stock selecionado pelo cliente. Se a quantidade escolhida pelo cliente não ultrapassar a definida na base de dados o sistema não age, caso contrário atualiza a quantidade escolhida pelo cliente para a máxima aceite pela base de dados.

CWE-209: Generation of Error Message Containing Sensitive Information

A vulnerabilidade CWE-209, conhecida como "Generation of Error Message Containing Sensitive Information", refere-se a um problema de segurança comum em sistemas de software e aplicativos que ocorre quando informações sensíveis são expostas indevidamente através de mensagens de erro. No nosso sistema, esta vulnerabilidade está presente na app insegura quando um usuário realiza uma busca que não retorna resultados, como a procura de um produto ou de uma categoria que não exista. Nestas situações, o sistema exibe mensagens de erro que contêm informações diretamente relacionadas à base de dados.

Estas informações podem ser exploradas por um potencial atacante, uma vez que fornecem insights sobre a estrutura do sistema e possíveis fraquezas. Portanto, foi crucial abordar essa vulnerabilidade para demonstrar a não proteção de dados sensíveis e a falta de integridade de um sistema.

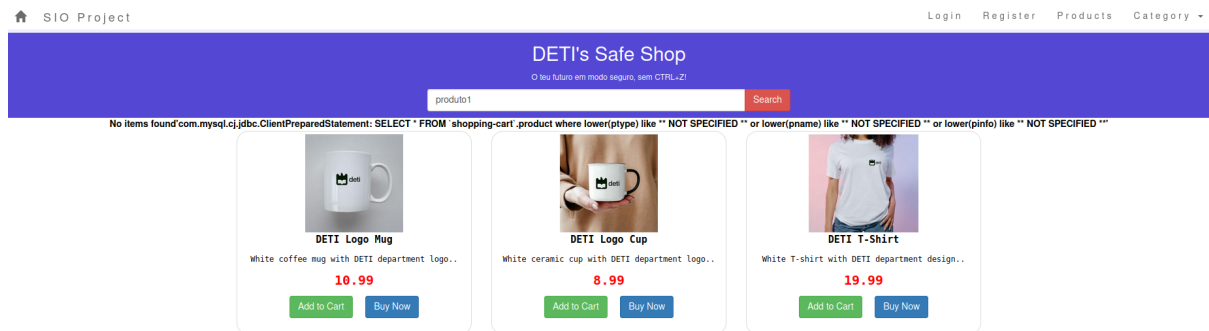


Figura 26

O grupo identificou que esta vulnerabilidade era decorrente de uma prática inadequada de programação. No sistema, o objeto estava a ser exibido diretamente, em vez de ser apresentada uma mensagem personalizada ou utilizar um método como ToString().

```
53     if (products.isEmpty()) {  
54  
55         message = "No items found" + prodDao.ps + "";  
56         products = prodDao.getAllProducts();  
57  
58     }
```

Figura 27

Para a resolução desta vulnerabilidade, o grupo realizou uma melhoria no código, passando a exibir mensagens personalizadas com base no valor inserido pelo usuário na barra de pesquisa. Essa abordagem permite comunicar de forma mais clara e amigável com o utilizador, sem expor informações sensíveis ou detalhes técnicos do sistema. Essa prática não apenas aumenta a segurança, mas também melhora a experiência do utilizador ao fornecer feedback mais informativo e relevante.

```
56     if (products.isEmpty()) {
57         message = "No items found for the search '" + (escapedSearch != null ? escapedSearch : type) + "'";
58         products = prodDao.getAllProducts();
59     }
```

Figura 28 a)

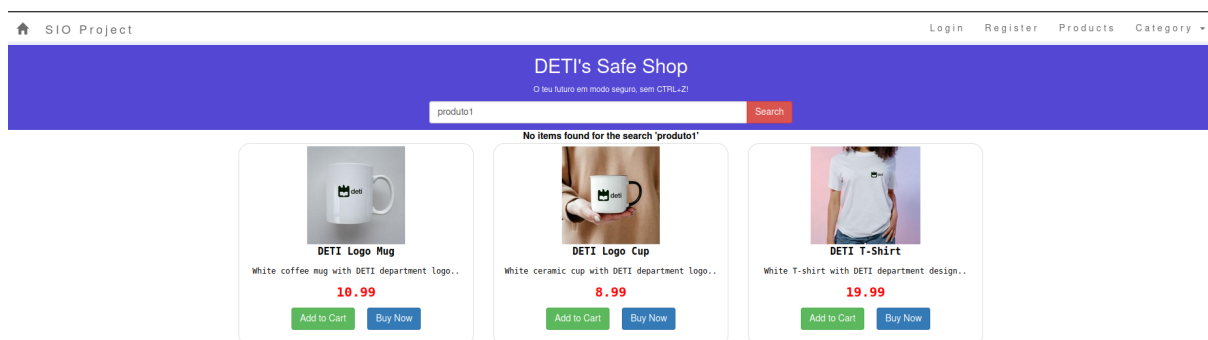


Figura 28 b)

CWE-1284: Improper Validation of Specified Quantity in Input

A CWE-1284: Improper Validation of Specified Quantity in Input refere-se, como indica o nome, à falha do sistema em validar corretamente uma quantidade específica dada como input. Isto pode resultar em comportamentos inesperados por parte do sistema pois irá estar a trabalhar sob valores não esperados.

Um exemplo prático desta falha no nosso sistema seria um cliente efetuar uma compra de 2 itens, sendo um deles com quantidade negativa e outro com quantidade positiva com o objetivo de a soma total dos valores seja 0. Desta forma o cliente paga 0 e efetua uma encomenda.

Cart Items



Picture	Products	Price	Quantity	Add	Remove	Amount
	DETI T-Shirt	19.99	<input type="text" value="1"/> <input type="button" value="Update"/>	+	-	19.99
	DETI T-Shirt	19.99	<input type="text" value="-1"/> <input type="button" value="Update"/>	+	-	-19.99
Total Amount to Pay (in Euros)						0.0
<input type="button" value="Cancel"/> <input type="button" value="Pay Now"/>						

Figura 29

Para resolver este problema, temos de garantir que o input enviado pelo utilizador nunca é abaixo de zero, para isso basta utilizarmos o formulário html, para ter um valor mínimo igual a zero.

```
<td><form method="post" action="./UpdateToCart">
    <input type="number" name="pqty" value="<%=prodQuantity%>"
        style="max-width: 70px;" min="0"> <input type="hidden"
        name="pid" value="<%=product.getProdId()%>"> <input
        type="submit" name="Update" value="Update"
        style="max-width: 80px;">
</form></td>
```

Figura 30

Também podemos garantir que caso a conta final seja zero, o sistema não disponibiliza a opção de efetuar a compra.

```
<%
if (totAmount != 0) {
%>
<tr style="background-color: #grey; color: #white;">
<td colspan="4" style="text-align: center;">
<td><form method="post">
    <button formaction="userHome.jsp"
        style="background-color: #black; color: #white;">Cancel</button>
</form></td>
<td colspan="2" align="center"><form method="post">
    <button style="background-color: #blue; color: #white;"
        formaction="payment.jsp?amount=<%=totAmount%>">Pay Now</button>
</form></td>
```

Figura 31

Conclusão

A finalidade deste projeto consistia em investigar e mitigar potenciais vulnerabilidades em projetos de software. Durante a execução deste trabalho, conseguimos aprimorar a nossa compreensão da parte teórica sobre as vulnerabilidades, compreendendo o que são, como operam e como evitá-las em futuros projetos. Em termos práticos, desenvolvemos o nosso conhecimento sobre algumas ferramentas já utilizadas em algumas cadeiras, e aprendemos sobre linguagens e serviços que não tínhamos utilizado previamente, pelo que foi uma experiência enriquecedora.

Em resumo, consideramos que alcançamos eficazmente o objetivo principal de destacar a importância da utilização de sistemas seguros no nosso projeto.

Referências

CWE - 2023 CWE Top 25 Most Dangerous Software Weaknesses. (n.d.). Cwe.mitre.org.
https://cwe.mitre.org/top25/archive/2023/2023_stubborn_weaknesses.html

MITRE. (n.d.). History of Common Weakness Enumeration (CWE). Cwe.mitre.org.
<https://cwe.mitre.org/about/history.html>

Security-Database. (n.d.). Security-Database.
<https://www.security-database.com/>

Perforce. (n.d.). What Is CWE? | Perforce.
<https://www.perforce.com/blog/kw/what-is-cwe>