



deti

universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Segurança Informática e nas Organizações

Relatório Projeto 1

Vulnerabilities in Software Products

Nome: Diogo Almeida N°Mec: 108902
Nome: Rodrigo Aguiar N°Mec: 108969
Nome: Gonçalo Ferreira N°Mec: 107853
Nome: Tomás Matos N°Mec: 108624

Introdução	3
Implementação	4
Vulnerabilidades	13
Cross Site Scripting (CWE-79)	14
SQL Injection (CWE-89)	19
Improper Input Validation (CWE-20)	21
Missing Authorization / Exposure Of Sensitive Information to an Unauthorised Actor (CWE - 862 / CWE-200)	24
Improper Authentication / Missing Authentication for Critical Function (CWE - 287 / CWE-306)	28
Use of Hard Coded Credentials / Missing Encryption of Sensitive Data / Plaintext storage of a Password (CWE-798 / CWE-311 / CWE - 256)	31
Unverified Password Change (CWE-620)	33
Weak Password Requirements / Use of Weak Credentials (CWE-521) (CWE-1391)	38
Inclusion of Functionality from Untrusted Control Sphere (CWE - 829)	39
Generation of Error Messages Containing Sensitive Information (CWE - 209)	40
Conclusão	41
Bibliografia	42

Introdução

O projeto de desenvolvimento da Loja Online do DETI é uma iniciativa na qual trabalhamos para criar um ambiente de aprendizagem prática para nós. O nosso principal objetivo neste projeto foi a importância da segurança em aplicações de software enquanto desenvolvemos uma loja online funcional que disponibiliza produtos relacionados com o nosso departamento.

A nossa abordagem única envolve a introdução controlada de vulnerabilidades que não são óbvias para o utilizador comum. Temos a responsabilidade de desenvolver tanto uma versão com falhas como uma versão corrigida da loja online. Além disso, iremos documentar detalhadamente como essas vulnerabilidades são exploradas e qual o seu impacto.

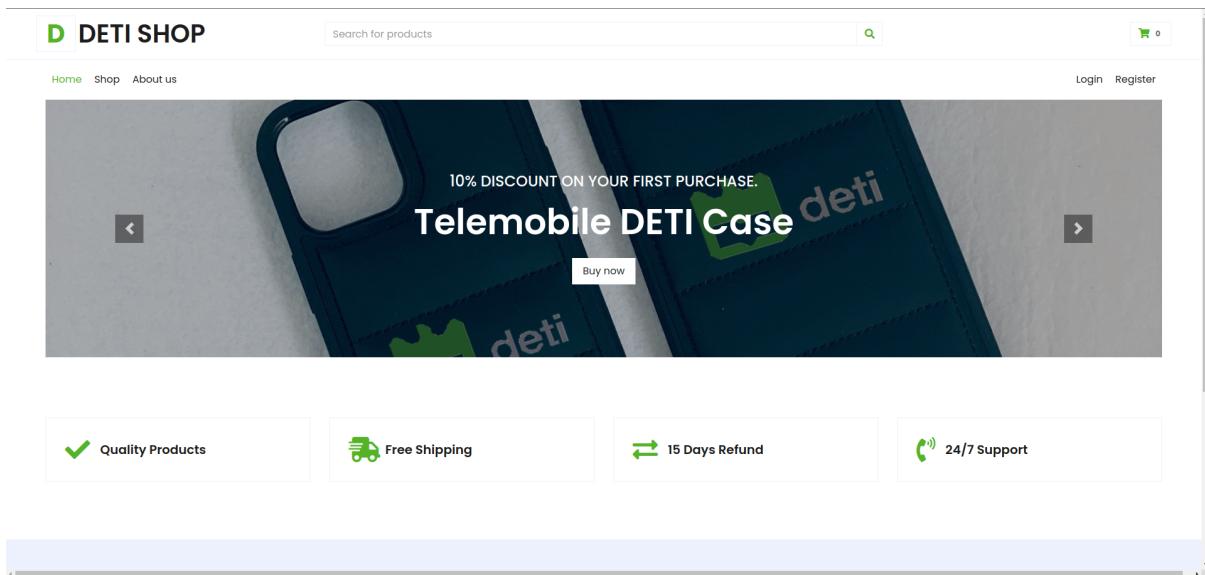
Neste projeto, concentramo-nos na compreensão da importância da segurança de software, na identificação de vulnerabilidades comuns, como Cross-Site Scripting e Injeção SQL, e na aquisição de conhecimento sobre como as corrigir.

Este relatório irá abordar os principais aspectos do projeto, incluindo o propósito da loja online, as vulnerabilidades introduzidas por nós, as explorações e o seu impacto, bem como as estratégias que desenvolvemos para evitar essas vulnerabilidades. Além disso, iremos discutir a implementação do nosso código.

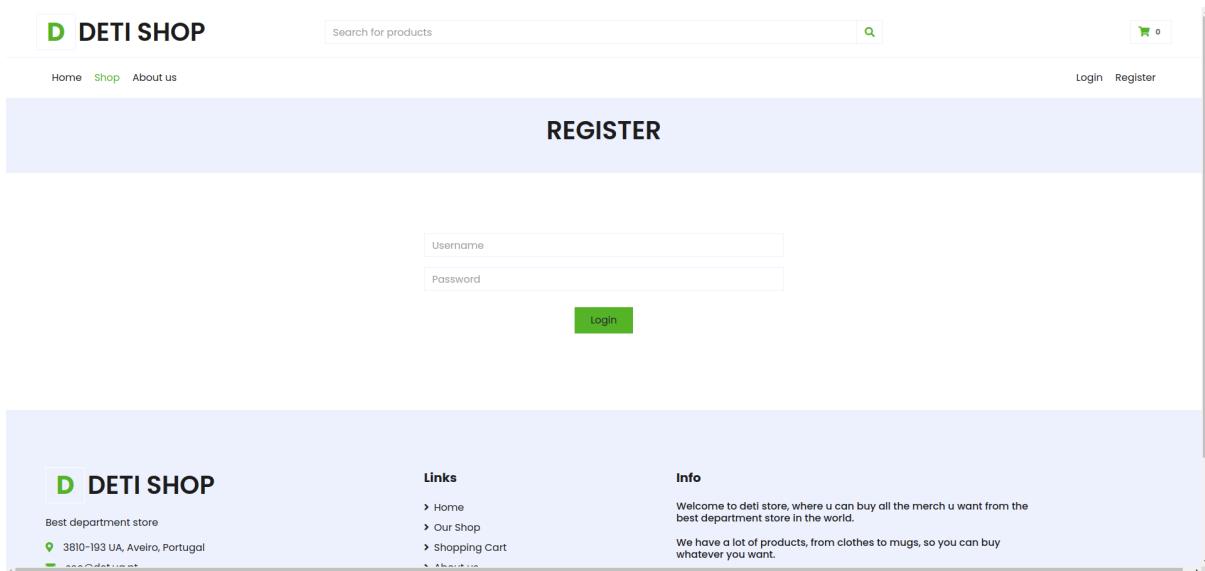
O nosso projeto não se limita apenas à criação de uma aplicação funcional, mas também à capacitação de todos nós na compreensão e correção de vulnerabilidades de segurança. Isso vai preparar-nos para enfrentar os desafios do desenvolvimento de software num ambiente cada vez mais focado na segurança.

Implementação

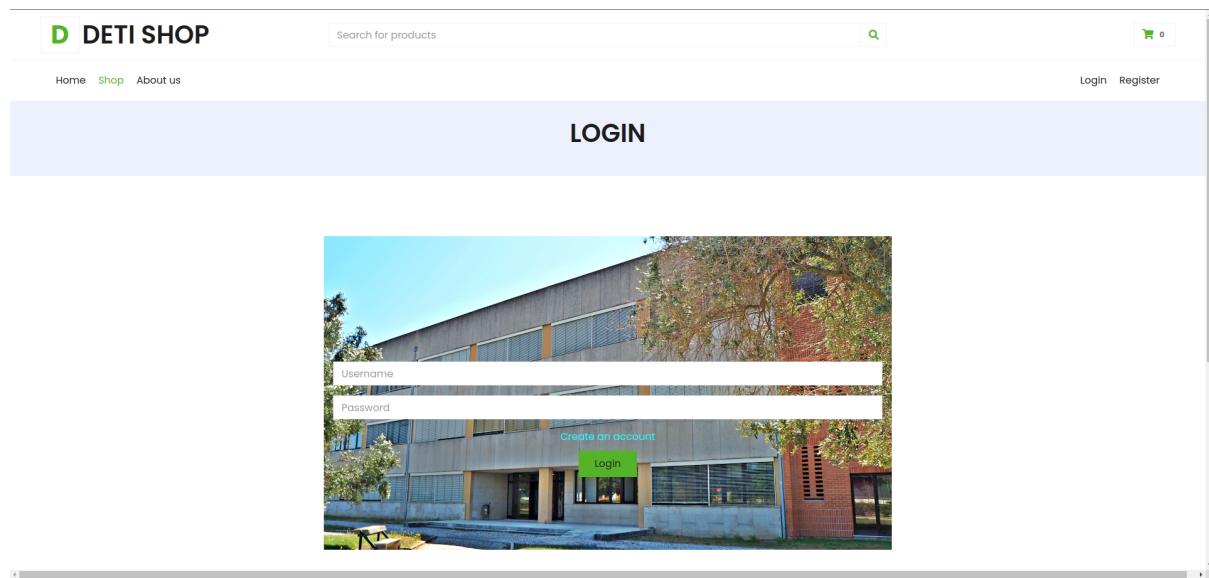
Quando entra no nosso site vai se deparar com a página inicial da nossa loja do Deti:



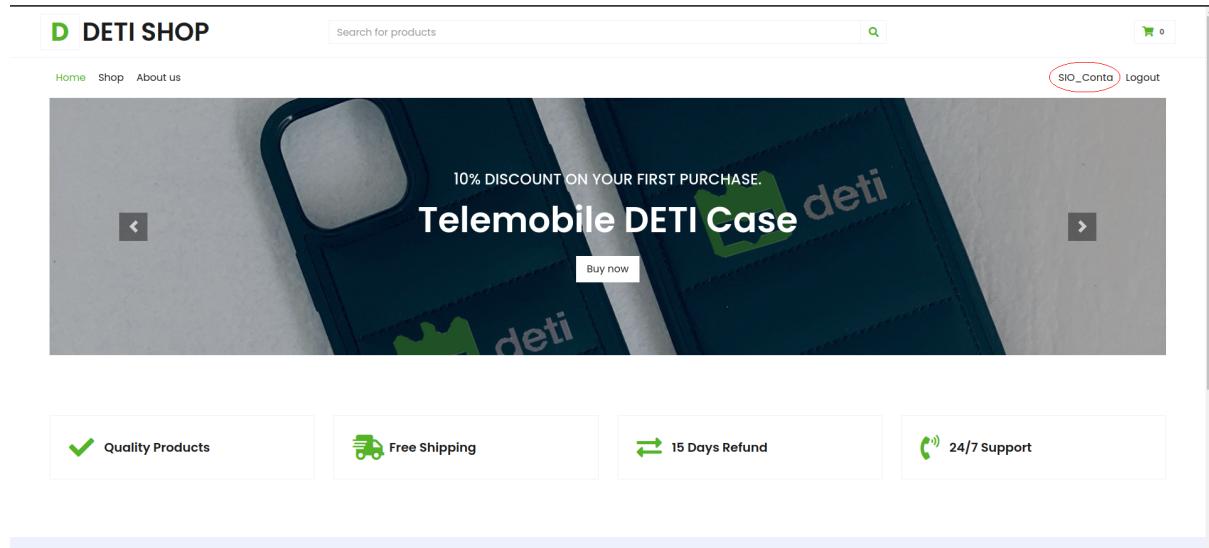
Aqui poderá criar uma conta clicando no botão Register e irá ser redirecionado para a página com os campos para fazer a respetiva criação de conta:



Depois de colocar os seus dados a conta será criado e irá ser redirecionado para a página inicial novamente onde poderá então clicar no botão Login e entrar na sua conta:



Caso os dados introduzidos sejam de uma conta válida irá ser redirecionado para a página inicial mas com a indicação que está na sua conta (assinalado na figura):



Agora ao clicar no botão Shop vai para a lista de produtos da nossa loja onde poderá utilizar o filtro de forma a exibir apenas os produtos a gosto:

OUR SHOP

Home - Shop

Filter by price

All Price
 \$0 - \$10
 \$10 - \$20
 \$20 - \$30
 \$30 - \$40
 \$40 - \$50

Search by name

 Caneca do Deti Price: 15.00€ View Detail Add To Cart	 Pin do Deti Price: 5.00€ View Detail Add To Cart	 T-shirt do Deti Price: 20.00€ View Detail Add To Cart
		

Com filtro:

OUR SHOP

Home - Shop

Filter by price

All Price
 \$0 - \$10
 \$10 - \$20
 \$20 - \$30
 \$30 - \$40
 \$40 - \$50

Search by name

 Pin do Deti Price: 5.00€ View Detail Add To Cart


Agora na Shop pode ver mais detalhes de um produto clicando no botão View Details no produto que desejar e aí irá encontrar uma breve descrição e também uma secção de comentários onde os clientes podem deixar uma review dos produtos:

The screenshot shows a product detail page for a mug. At the top, there's a large image of the mug on a surface. To its right, the product title "Caneca do Deti" and price "15€" are displayed. Below the title is a brief description: "Ganda caneca para representar o departamento de informática (e do resto) e pa beber café no aquário". Underneath the description are quantity selection buttons (-, 1, +) and an "Add To Cart" button. Below these buttons is a navigation bar with tabs for "Description" and "Reviews", where "Reviews" is currently selected. On the left side of the reviews section, there's a heading "1 reviews for Caneca%20do%20Deti" followed by a review from "SIO_Conta - 2-II-2023" which reads "Que bela caneca :)".

Agora poderá adicionar vários produtos ao carrinho selecionando a quantidade e depois clicando em Add To Cart ou então na lista de produtos clicando no Add to Cart. O número de produtos vai aumentando no ícone do carrinho de forma a dar uma vista mais interativa ao cliente. Neste caso irei adicionar 5 canecas ao carrinho:

The screenshot shows the shop detail page for the "Caneca do Deti". The main content area features the product image, title, price, and description. Below the description is the quantity selector with the number "5" highlighted by a red oval. To the right of the selector is the "Add To Cart" button. The top navigation bar includes the shop logo, a search bar, and user information for "SIO_Conta". The bottom of the page has a footer with links for "Home", "Shop", "About us", "Logout", and a copyright notice.

Agora ao clicar no ícone do carrinho irá para o seu carrinho de compras e verá todos os produtos selecionados. Aqui poderá editar a quantidade de produtos ou então proceder para a compra dos mesmos clicando em Proceed to Checkout:

Products	Price	Quantity	Total	Remove
Caneca do Deti	15€	5	75€	

Cart Summary	
Subtotal	75€
Shipping	10€
Total	85€

[Proceed To Checkout](#)

Order Total	
Products	Caneca do Deti x 5 75.00€
Subtotal	75.00€
Shipping	10€
Total	85.00€

Payment	
<input type="radio"/> Paypal	
<input type="radio"/> Credit card	

[Place Order](#) 8

Information	
First Name	Last Name
John	Doe
E-mail	Mobile No
example@email.com	+123 456 789
Shipping Address	Country
123 Street	Portugal
City	ZIP Code
New York	123

Order Notes	
Notes about your order, e.g. special notes for delivery	

Agora basta preencher os dados de faturação, selecionar a forma de pagamento e clicar em Place Order para enviar a compra para ser preparada:

Information

First Name	Last Name
SIO	Cliente
E-mail	Mobile No
sio_email@gmail.com	998877665
Shipping Address	Country
Rua 1, 1234, Aveiro	Portugal
City	ZIP Code
Aveiro	1234-123

Order Notes

Encomenda muito urgente.

Order Total

Products	
Caneca do Deti x 5	75.00€
Subtotal	75.00€
Shipping	10€
Total	85.00€

Payment

Paypal
 Credit card

Depois de enviar o pedido será redirecionado para a página inicial. Agora poderá clicar no seu Username no canto superior direito e verá todas as suas informações incluindo os pedidos já realizados:

Home Shop About us
SIO_Conta Logout

USER DETAIL

User: SIO_Conta

[Change password:](#)

[Reset](#)

Reset password to a random one:

Orders:

Firstname	Lastname	Email	Ship Address	Country	City	ZipCode	Username	Products Info	Total Price	Payment Method
SIO	Cliente	sio_email@gmail.com	Rua 1, 1234, Aveiro	Portugal	Aveiro	1234-123	SIO_Conta	View Details	85€	creditcard

Como se pode ver já está ali o pedido realizado. Clicando em View Details poderá ver-se as informações do pedido:

The screenshot shows a user detail page with a modal window titled "Order Information". Inside the modal, it says "Product: Caneca do Deti, Quantity: 5". Below the modal, there's a section titled "User: SIO_Conta" with "Change password:" and "Reset password to a random one:" buttons. At the bottom, there's a table titled "Orders:" with columns: Firstname, Lastname, Email, Ship Address, Country, City, ZipCode, Username, Products Info, Total Price, and Payment Method. One row is shown: SIO, Cliente, sio_email@gmail.com, Rua 1, 1234, Aveiro, Portugal, Aveiro, 1234-123, SIO_Conta, [View Details](#), 85€, creditcard.

Nesta página também poderá alterar a password ou então reset para uma password random mais segura.

Agora caso façamos login com as credenciais de Administrador somos redirecionados para a página de administrador:

The screenshot shows the administrator panel with a header "DETI SHOP" and "Admin Panel". Below is a "Order List" section with three items: "Caneca do Deti" (Price: \$15, Stock: 31 units), "Pin do Deti" (Price: \$5, Stock: 60 units), and "T-shirt do Deti" (Price: \$20, Stock: 53 units). Each item has "Change Stock" and "Change Price" buttons. There are also two additional items partially visible below: a white t-shirt and two blue phone cases.

Aqui o/os administrador/es poderão consultar as unidades de cada produto em stock podendo adicionar mais unidades ao stock ou então alterar o preço dos produtos:



Pin do Deti

Price: \$5

Ganda pin para representar o departamento de informática (e do resto) e pa meter na mochila

Stock: 60 units

Change Stock **Change Price**

localhost:8000 diz

Enter new stock value for Pin do Deti:

Cancelar **OK**



Pin do Deti

Price: \$5

Ganda pin para representar o departamento de informática (e do resto) e pa meter na mochila

Stock: 100 units

Change Stock **Change Price**

Mais abaixo encontramos os pedidos ativos, neste caso está aqui o pedido que realizámos anteriormente:

The screenshot shows a table with two columns: OrderID and Username. The OrderID row contains the value 1660AC2C180ECEF3. The Username row contains the value SIO_Conta. Below this table is a modal window with the following details:

Order ID: 1660AC2C180ECEF3
Username: SIO_Conta
First Name: SIO
Last Name: Cliente
Email: sio_email@gmail.com
Phone Number: 998877665
Shipping Address: Rua 1, 1234, Aveiro
Country: Portugal
City: Aveiro
Zip Code: 1234-123
Products Info: [{"product": "Caneca do Deli", "quantity": 5}]]

At the bottom of the modal are two buttons: View Details and Update.

Caso seja necessário também é possível dar Update às informações dos pedidos clicando em Update no pedido:

The screenshot shows a table with two columns: OrderID and Username. The OrderID row contains the value 1660AC2C180ECEF3. The Username row contains the value SIO_Conta. Below this table is a modal window with the following fields:

SIO	Cliente	sio_email@gmail.com	998877665
Rua 1, 1234, Aveiro	Portugal	Aveiro	1234-123
[{"product": "Caneca do Deli", "quantity": 5}]	<input type="button" value="Update"/>		

Vulnerabilidades

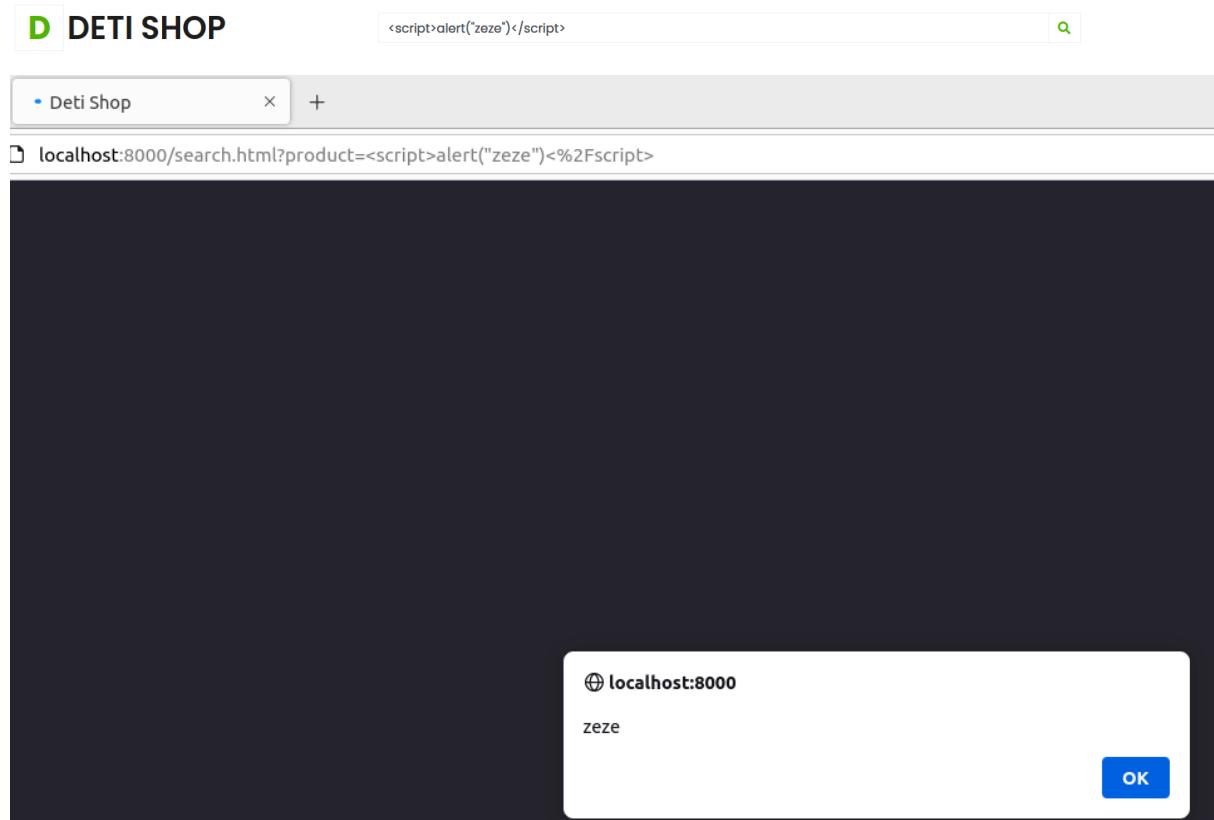
CVE Number	Descrição	CWE - Related	Página
1	Cross Site Scripting https://cwe.mitre.org/data/definitions/79.html	CWE-79	Página 14
2	SQL Injection https://cwe.mitre.org/data/definitions/89.html	CWE-89	Página 19
3	Improper Input Validation https://cwe.mitre.org/data/definitions/20.html	CWE-20	Página 21
4	Missing Authorization https://cwe.mitre.org/data/definitions/862.html	CWE-862	Página 24
5	Exposure of sensitive information to an unauthorised actor https://cwe.mitre.org/data/definitions/200.html	CWE-200	Página 24
6	Improper Authentication https://cwe.mitre.org/data/definitions/287.html	CWE-287	Página 28
7	Missing authentication for critical function https://cwe.mitre.org/data/definitions/306.html	CWE-306	Página 28
8	Use of hard-coded credentials https://cwe.mitre.org/data/definitions/798.html	CWE-798	Página 31
9	Unverified Password Change https://cwe.mitre.org/data/definitions/620.html	CWE-620	Página 33
10	Weak Password Requirements https://cwe.mitre.org/data/definitions/1391.html	CWE-1391	Página 38
11	Plaintext Storage of Password https://cwe.mitre.org/data/definitions/256.html	CWE-256	Página 31
12	Use of Weak Credentials https://cwe.mitre.org/data/definitions/521.html	CWE-521	Página 38
13	Inclusion of Functionality from Untrusted Control Sphere https://cwe.mitre.org/data/definitions/829.html	CWE-829	Página 39
14	Generation of Error Messages Containing Sensitive Information https://cwe.mitre.org/data/definitions/209.html	CWE-209	Página 40

Cross Site Scripting (CWE-79)

O XSS, Cross-Site Scripting, acontece quando não se consegue lidar corretamente com os inputs não confiáveis fornecidos pelos utilizadores. Podem acontecer a partir de um pedido web (formulário ou URL) quando a aplicação gera uma outra página web com esses dados não confiáveis sem qualquer validação para algum perigo. Ao visitar esta nova página, o input malicioso que o utilizador inseriu é executado, a partir desta vulnerabilidade o atacante pode roubar informações privadas, assumir controlo do computador da vítima,

- Reflected XSS (Não-Persistente) - O código malicioso é refletido na resposta do servidor para o cliente quando um atacante fornece um URL perigoso à vítima, comum em esquemas de phishing.
- Stored XSS (Persistente) - A aplicação armazena esses dados perigosos na base de dados que podem ser exibidos para outros utilizadores.
- DOM-Based XSS - O código maligno é injetado no lado do cliente, este XSS envolve um script controlado pelo servidor que é enviado para o cliente. No caso deste script processar dados fornecidos pelo atacante e os integrar na página web esta vulnerabilidade existirá.

Um exemplo de esta vulnerabilidade é a search bar dos produtos que existem na loja:

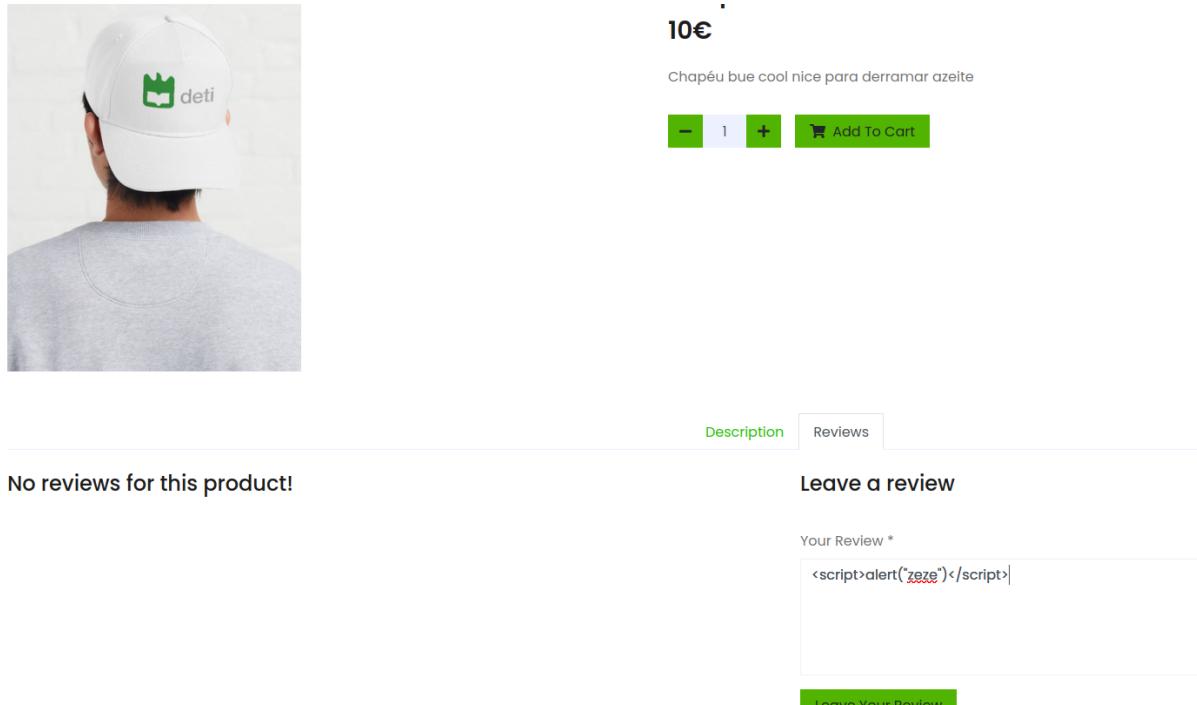


Como podemos ver no código abaixo, o nome do produto é diretamente retirado do link e, sem nenhuma filtragem, é adicionado ao website, através da função `html()` do jquery, que é insegura, pois não altera a componente de texto do elemento mas sim adiciona diretamente o nome do produto ao elemento.

```
<script>
    const urlParams = new URLSearchParams(window.location.search);
    const productName = urlParams.get("product");
    $("#product_name").html(productName);
```

A nossa aplicação insegura também apresenta esta vulnerabilidade nas reviews dos produtos, desta vez é um XSS do tipo Stored XSS, uma vez que existindo uma review com um script, este será executado sempre que um utilizador aceder à página dos product details onde se encontra a review maliciosa, foi usada

local storage para guardar as reviews, mas também poderia estar armazenada numa tabela da base de dados:



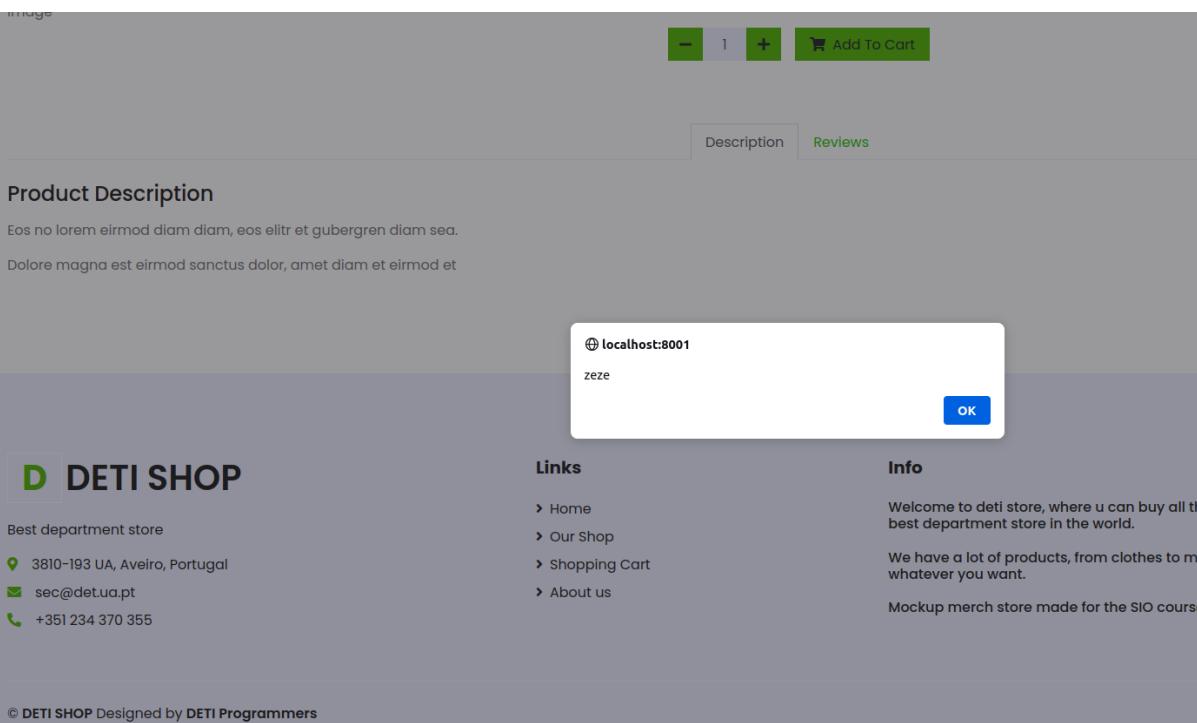
No reviews for this product!

Leave a review

Your Review *

```
<script>alert("zeze")</script>
```

Leave Your Review



localhost:8001

zeze

OK

D DETI SHOP

Best department store

3810-193 UA, Aveiro, Portugal

sec@deti.ua.pt

+351 234 370 355

Links

- Home
- Our Shop
- Shopping Cart
- About us

Info

Welcome to deti store, where u can buy all the best department store in the world.

We have a lot of products, from clothes to whatever you want.

Mockup merch store made for the SIO course

© DETI SHOP Designed by DETI Programmers

Template by HTML Codex

Código demonstrativo, função que vai buscar as reviews:

```
function loadDetails() {  
    var url = window.location.search;  
    var id = url.split('=')[1];  
    console.log("URL" , id);  
  
    var reviews = JSON.parse(localStorage.getItem("reviews_" + id));
```

A forma como a review está a ser apresentada é desta forma, claramente quando um utilizador colocar como input algo do gênero <script></script> esse script será executado:

```
for (var i = 0; i < reviews.length; i++) {
    var review = reviews[i]
    //review = review.replace(/(?:\r\n|\r|\n)/g, '<br>');
    var reviewCard =
        '<div>
            <h5>
                ${review.user}<small> - <i>${review.date}</i></small>
            </h5>
            <p >
                ${review.review}
            </p>
        </div>
    ';
    $(reviewContainer).append(reviewCard);
}
$("#product_name").replaceWith('<h3 class="mb-4" id = "product_name" >' + review_count + ' reviews for ' + id + '</h3>')
```

Para corrigir estes problemas de XSS na aplicação segura reforçamos as validações de input, aplicando HTML Escaping em todos os campos que poderiam ser alterados pelo utilizador com o atributo .textContent do HTML, que por si próprio forneceu uma boa segurança contra ataques de XSS.

```
for (var i = 0; i < reviews.length; i++) {
    var review = reviews[i]
    //review = review.replace(/(?:\r\n|\r|\n)/g, '<br>');
    var reviewCard = document.createElement('div');

    var h5 = document.createElement('h5');
    h5.textContent = review.user + ' - ' + review.date ;
    reviewCard.appendChild(h5);

    var p = document.createElement('p');
    p.textContent = review.review;
    reviewCard.appendChild(p);

    $(reviewContainer).append(reviewCard);
}
$("#product_name").replaceWith('<h3 class="mb-4" id = "product_name" >' + review_count + ' reviews for ' + id + '</h3>')
```

Para testar a eficácia deste método, testámos estas caixas de texto com vários tipos de inputs presentes em [XSS Filter Evasion Cheat Sheet](#), como por exemplo:

```
javascript:/*--></title></style></textarea></script></xmp>
<svg/onload='+/" /+/onmouseover=1/+/[*/[ ]/+alert(1)//'>
```

- Input que testa XSS em várias vertentes (HTML, js, css)

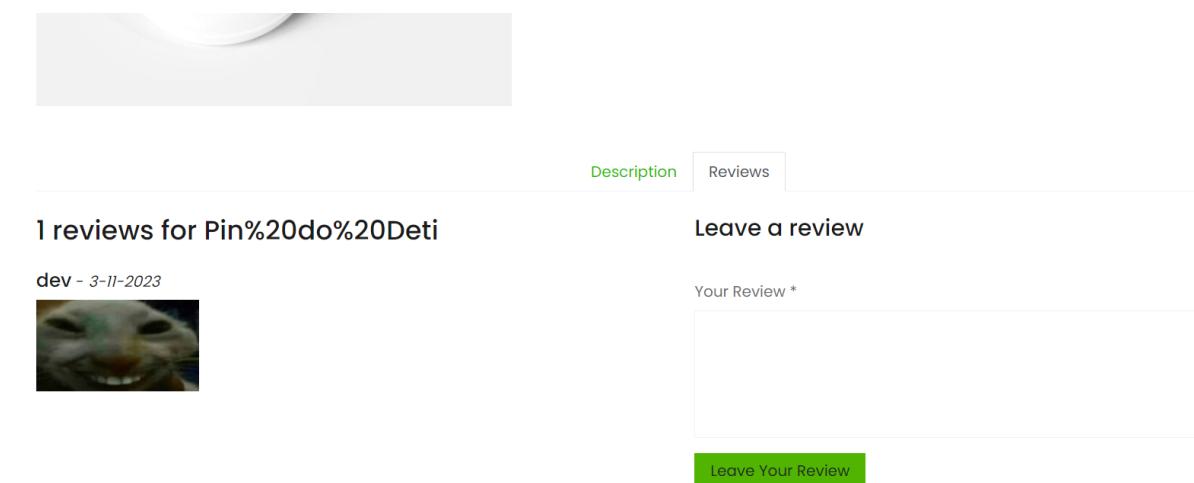
O website mostrou-se capaz de resistir a estes ataques, não executando as várias payloads de XSS testadas mas sim apenas mostrando-as como texto no website.

Mais detalhes desta técnica podem ser vistos na descrição da vulnerabilidade associada à [CWE-20](#).

A segunda barreira de defesa que implementamos para prevenir XSS foi um CSP restritivo.

Um CSP (Content Security Policy) consiste num conjunto de regras que especificam quais fontes de dados um servidor web pode utilizar.

Na aplicação insegura não foi implementado um CSP sendo que é possível aceder a conteúdos de qualquer outro website como por exemplo imagens:



Também é possível executar scripts/estilos/fontes de qualquer fonte ou fazer conexão do web server a qualquer outro sistema.

Um atacante poderá utilizar isto para, conjuntamente com XSS, fazer a exfiltração de dados e enviar para o seu próprio servidor.

Na **app segura** foi implementado um CSP o mais restritivo possível, impedindo por exemplo a execução de qualquer script-inline no website ou acesso a scripts de fontes desconhecidas.

```
self.send_header(  
    "Content-Security-Policy",  
    "default-src 'none';"  
    "script-src 'self' cdn.jsdelivr.net code.jquery.com stackpath.bootstrapcdn.com 'nonce-filter-nonce' 'nonce-searchnonce' 'nonce-getcartnonce'  
    "connect-src 'self' http://localhost:5000 ;"  
    "img-src 'self' data:;";  
    "style-src 'self' cdn.jsdelivr.net 'unsafe-inline' https://fonts.googleapis.com https://fonts.gstatic.com https://cdnjs.cloudflare.com;";  
    "font-src 'self' cdn.jsdelivr.net fonts.gstatic.com fonts.googleapis.com cdnjs.cloudflare.com;";  
)
```

Entraremos mais em detalhe sobre este CSP utilizado e a sua eficácia na secção sobre a [CWE-829](#).

SQL Injection (CWE-89)

SQL Injection consiste num ataque em que são manipuladas as queries que são feitas a uma base de dados.

Um programa é normalmente vulnerável a estes ataques quando trata input dado pelo utilizador como fiável e concatena estes inputs diretamente às queries realizadas à base dados (conhecido como SQL Dinâmico).

No website inseguro existem vários campos onde é possível realizar SQL Injection.

Um exemplo já bastante conhecido é tentar autenticar-se como um utilizador sem saber a sua password.

```
def login():
    try:
        data = request.get_json()
        username = data["username"]
        password = data["password"]

        conn = sqlite3.connect("LojaDeti.db")
        cursor = conn.cursor()

        query = (
            "SELECT * FROM Users WHERE username ='" +
            username +
            "' AND pass ='" +
            password +
            "'"
    )
```

Como podemos ver na imagem acima, o login na api utiliza SQL Dinâmico, ou seja, concatena diretamente o input que é fornecido pelo utilizador à query SQL.

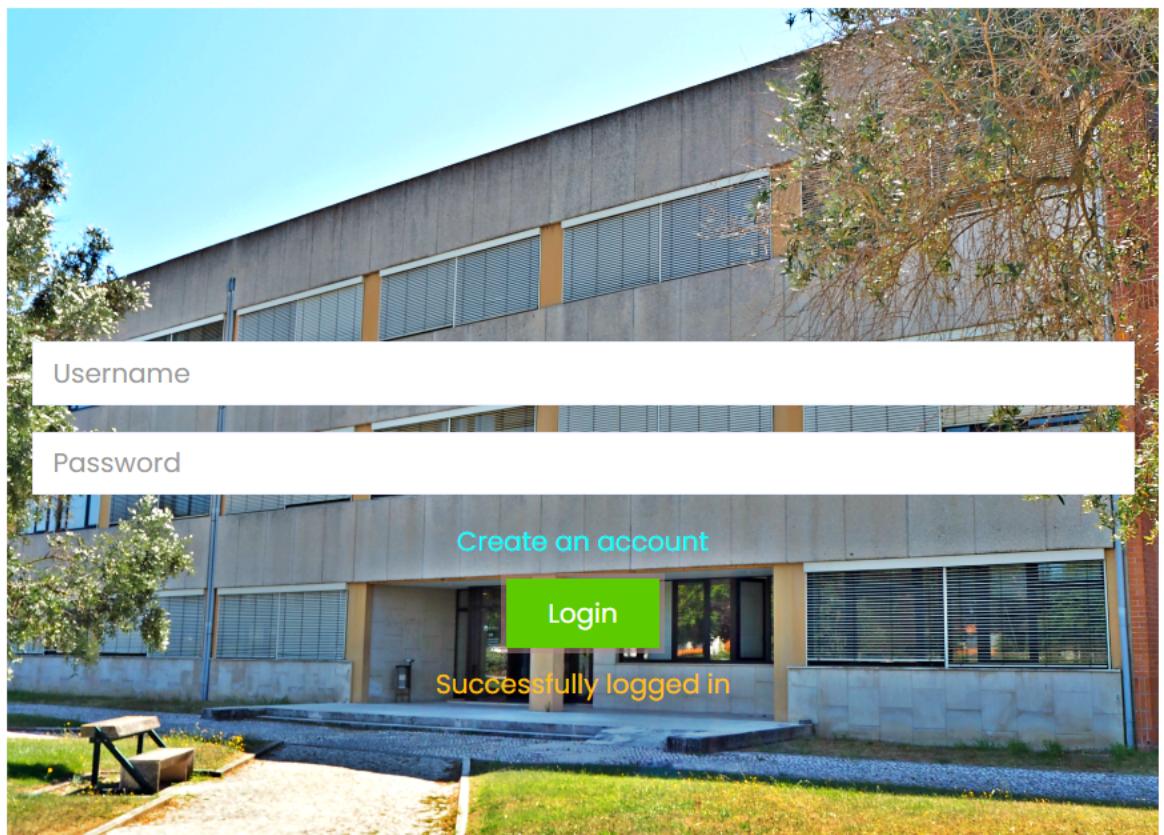
Um atacante pode utilizar isto para sua vantagem, inserindo estes campos no campo de registro:

Username: admin
Password: ' OR '1=1' --

Com este input será criada a seguinte query à base de dados:

```
Select * From Users WHERE username = 'admin' OR '1=1' --
```

Sendo a condição de **OR '1=1'** sempre verdadeira, o atacante consegue agora entrar no sistema como administrador.



Existem mais exemplos de campos em que é possível fazer SQL Injection, como por exemplo na barra de pesquisa, mas não é possível fazer queries com vários statements (como por exemplo terminar a query de select com ; e fazer um drop table), visto que a ferramenta utilizada para conexões à base de dados (SQLite 3) bloqueia este tipo de queries.

A maior defesa contra ataques de SQL Injection, a qual utilizamos na app segura, é o uso de queries sql parametrizadas, como podemos ver na imagem abaixo:

```

def register():
    try:
        data = request.get_json()
        username = data["username"]
        password = data["password"]
        hashed_password = generate_password_hash(password, method="pbkdf2:sha256")

        conn = sqlite3.connect("LojaDeti.db")
        cursor = conn.cursor()

        try:
            cursor.execute(
                "INSERT INTO Users VALUES (?, ?)", (username, hashed_password)
            )
        ...
    
```

Neste tipo de queries, são utilizados placeholders e os argumentos são apenas fornecidos em runtime, pelo que nunca são interpretados como SQL puro mas apenas inputs. Desta forma, a base de dados consegue fazer uma distinção entre código SQL e dados do utilizador, prevenindo que sejam feitas manipulações em SQL.

Improper Input Validation (CWE-20)

Fazer validação de input torna-se bastante importante num website pois é através de manipular inputs que um utilizador mal intencionado (hacker) consegue fazer Cross Site Scripting ou SQL Injection.

De entre as várias formas de validar input, a mais eficaz é conhecida por *HTML Escaping*.

Esta forma permite converter caracteres como '<' para a sua respectiva representação em HTML, neste caso '\$lt;', servindo como uma primeira barreira de defesa contra possíveis ataques de XSS.

Neste caso, na nossa implementação, utilizamos o atributo .textContent do HTML, caracterizado pelo [OWasp XSS Prevention CheatSheet](#) como um “Safe HTML Attribute”.

Este atributo automaticamente realiza HTML Escaping ao valor dado e adiciona-o como texto ao elemento HTML.

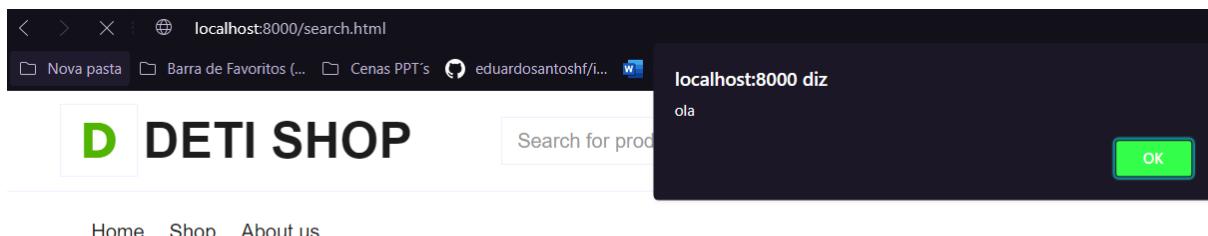
```
document.getElementById("product_name").textContent = productName;
```

Podemos comprovar a eficácia desta técnica, comparando o que acontece na app segura contra na app insegura quando se tenta inserir um input malicioso na barra de search do website.

App segura (textContent) :

The screenshot shows a web application interface. At the top left is a logo with a green 'D' and the text 'DETI SHOP'. To the right is a search bar with the placeholder 'Search for products'. Below the header, there is a navigation bar with links for 'Home', 'Shop', and 'About us'. The main content area has a heading 'Looking for:' followed by a large green-highlighted string of code: '<script>alert("ola")</script>'. Below this, the text 'Not Found' is displayed. At the bottom, there is a table header with columns labeled 'Name' and 'Price'.

App insegura (.html() do jQuery) :



Looking for:

Not Found

Name	Price

Podemos ver que na app segura o input do utilizador é tratado como texto e “escaped” de forma a que não seja executado como um script, como acontece na app insegura.

Também foram aplicadas outras formas de input validation no website, como por exemplo verificar inputs vazios ou se correspondiam a um certo padrão (regex) pretendido.

```
// Validation for Email - Ensure it's a valid email address
var emailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
if (!email.match(emailPattern)) {
    popup.textContent = "Input a valid email!";
    document.body.appendChild(popup);
    setTimeout(function () {
        document.getElementById("popup").style.display = "none";
    }, 2000)
    return;
}
```

Missing Authorization / Exposure Of Sensitive Information to an Unauthorised Actor (CWE - 862 / CWE-200)

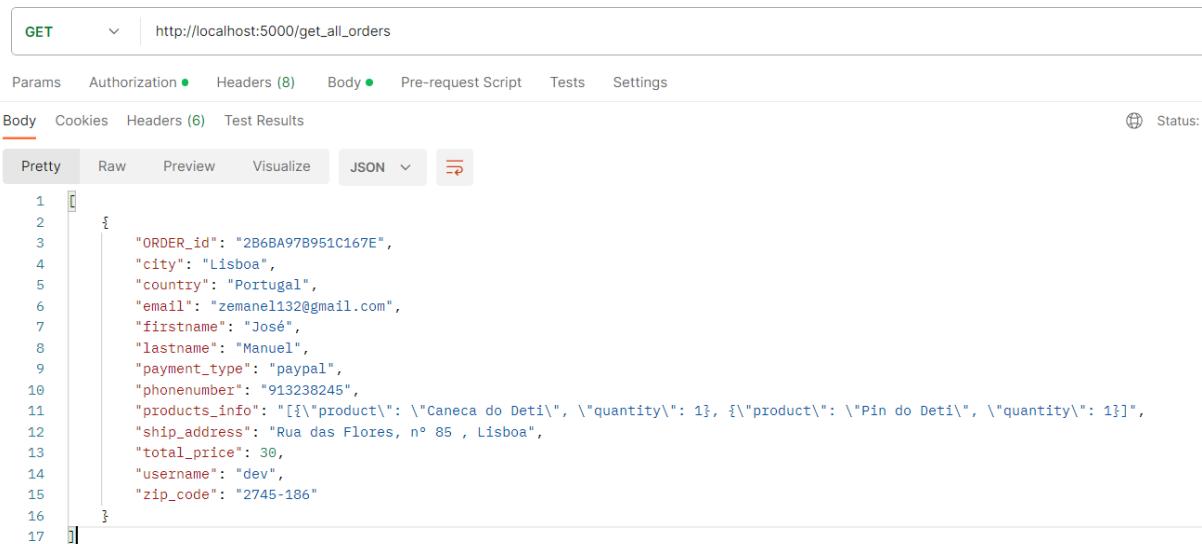
A Autorização (authorization) pode ser definida como verificar se um utilizador tem permissão para fazer uma determinada ação.

Havendo 2 entidades diferentes neste website, utilizadores e admin, é imperativo que por exemplo utilizadores não tenham acesso a recursos exclusivos para o administrador, como por exemplo ver todas as compras feitas no sistema ou alterar o preço de produtos.

No website seguro não está implementada autenticação na api, logo é extremamente fácil para um atacante alterar o preço dos produtos para 0 ou obter informações confidenciais sobre clientes, neste caso basta apenas saber o endereço ip , port e respetivo endpoint.

Como exemplo, mostramos algumas das ações que um atacante pode realizar na app insegura por falta de um sistema de autorização.

- Obter as informações de todas as compras no sistema (CWE 200)



The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: http://localhost:5000/get_all_orders
- Headers (8): Content-Type: application/json
- Body: Raw JSON response (Pretty)
- JSON Response:

```
1 {
2     "ORDER_id": "2B6BA97B951C167E",
3     "city": "Lisboa",
4     "country": "Portugal",
5     "email": "zemanel132@gmail.com",
6     "firstname": "José",
7     "lastname": "Manuel",
8     "payment_type": "paypal",
9     "phonenumber": "913238245",
10    "products_info": "[{\\"product\\": \"Caneca do Deti\", \\"quantity\\": 1}, {\\"product\\": \"Pin do Deti\", \\"quantity\\": 1}]",
11    "ship_address": "Rua das Flores, nº 85 , Lisboa",
12    "total_price": 30,
13    "username": "dev",
14    "zip_code": "2745-186"
15 }
16
17 }
```

- Alterar o preço de um produto

The screenshot shows a POSTMAN interface. At the top, it says "PUT" and the URL "http://localhost:5000/update_price/Caneca%20do%20Deti". Below the URL, there are tabs for "Params", "Authorization", "Headers (8)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is selected, showing options for "none", "form-data", "x-www-form-urlencoded", "raw", "binary", and "GraphQL". The "JSON" option is selected. The "Body" field contains the following JSON:

```
1 {"newPrice":0}
```

Below the body, there are tabs for "Body", "Cookies", "Headers (6)", and "Test Results". The "Body" tab is selected, showing options for "Pretty", "Raw", "Preview", "Visualize", and "JSON". The "Pretty" option is selected. The response body is displayed as:

```
1 {
2   "message": "Price updated successfully"
3 }
```

Estes são apenas alguns pequenos exemplos do que é possível fazer, sendo que seria também possível realizar SQL Injection, entre outros ataques, acedendo a outros endpoints da API.

Na app **segura** implementámos um sistema de autenticação e autorização com JSON Web Tokens (JWT's) .

Através do módulo `flask_jwt_extended` podemos definir autenticação e autorização em server-side com base nestes tokens.

```

if user and check_password_hash(user[1], password): # check the password
    if user[0] == "admin":
        access_token = create_access_token(
            identity=username,
            additional_claims={"role": "admin"},
            expires_delta=timedelta(minutes=30)
        )
        # print("Admin Logged in")
    else:
        access_token = create_access_token(
            identity=username, additional_claims={"role": "user"},
            expires_delta=timedelta(minutes=30)
        )
    # print("User " + username + " Logged in")

```

Como podemos ver na imagem acima, cada vez que um utilizador faz login, é criado um access token (JWT), sendo que a este token está associada uma “role” que pode ser “user” ou “admin”. Este token é posteriormente passado para o frontend e guardado em localStorage.

Nesta implementação, requests de http a endpoints protegidos da api são verificadas.

- É necessário um header Authorization com o respectivo token.

```

@jwt_required()
@app.route('/update_price/<product_name>', methods=['PUT'])
def update_price(product_name):

```

- É necessário ter permissão para aceder ao endpoint da api, neste caso ter a role certa e o utilizador que fez a request condizer com o dono do token.

```

def update_price(product_name):
    claims = verify_jwt_in_request()
    if "admin" == claims[1]["role"] and claims[1]["sub"] == get_jwt_identity():
        pass
    else:
        return Response(status=401, response=json.dumps({"error": "Unauthorized"}))
    try:

```

Testando novamente o ataque anterior de alterar o preço de um produto para 0, o que acontece é o seguinte:

The screenshot shows a POST request to `http://localhost:5000/update_price/Caneca%20do%20Deti`. The **Authorization** tab is selected, showing a dropdown set to **Bearer Token**. The response body is a JSON object with one item: `"msg": "Missing Authorization Header"`.

Apenas com um token de admin é possível realizar esta operação:

The screenshot shows a POST request to `http://localhost:5000/update_price/Caneca%20do%20Deti`. The **Authorization** tab is selected, showing a dropdown set to **Bearer Token** with a red box around it. The **Token** field contains a long string of characters. The response body is a JSON object with one item: `"message": "Price updated successfully"`.

Tentar fazer o mesmo só que com um token com role “user” irá dar erro 401 Unauthorized.

Improper Authentication / Missing Authentication for Critical Function (CWE - 287 / CWE-306)

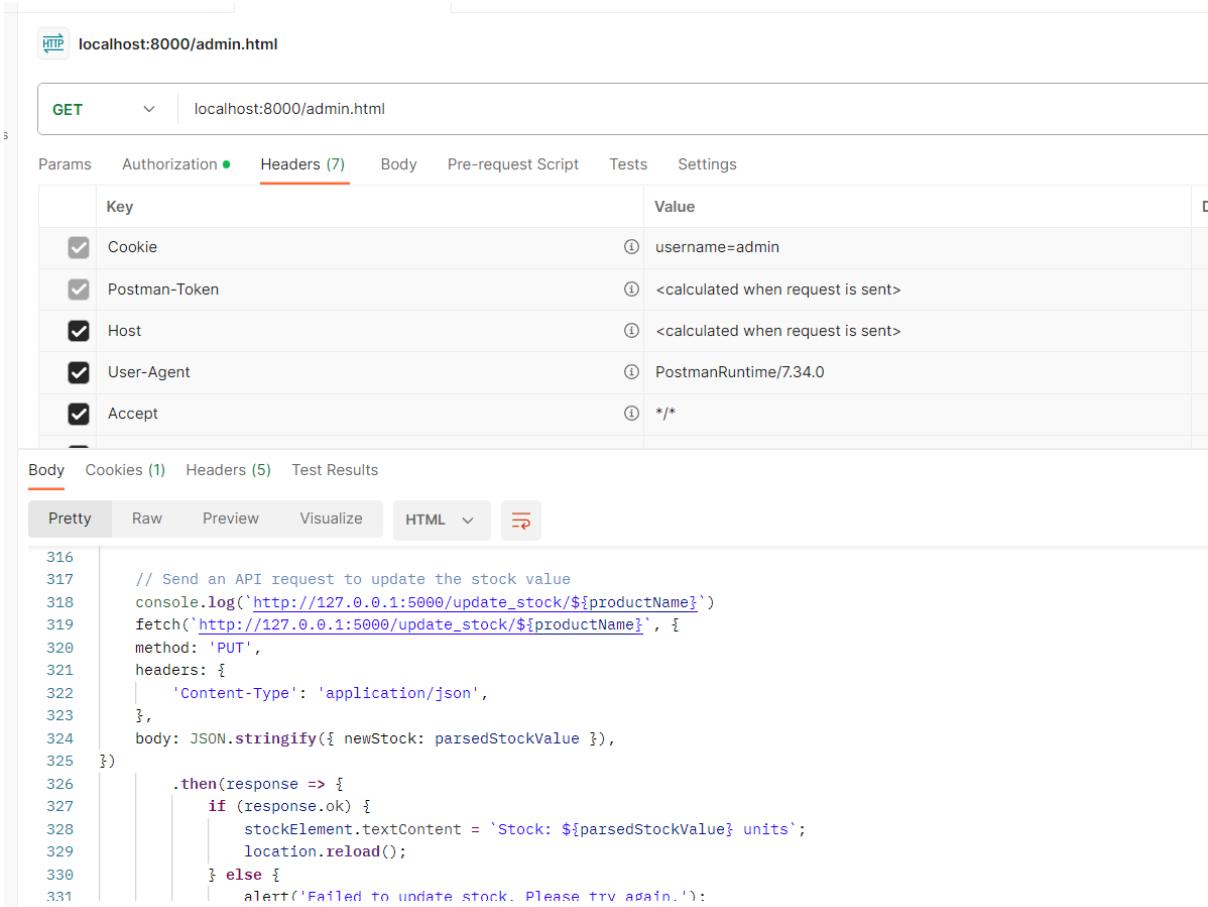
A autenticação consiste no processo de identificar um utilizador no sistema.

Na app insegura, a autenticação é feita client-side, através de um cookie com o valor username = \$nome.

Este cookie é atribuído quando um utilizador faz login e têm uma duração de 30 dias.

Sendo que esta é a única forma de autenticar um utilizador, é fácil para um atacante fazer-se passar pelo utilizador ou até mesmo roubar a sua sessão.

Fazendo-se passar por admin, utilizando um cookie com username = admin, um utilizador pode até obter acesso a todo o source-code da página de admin.



The screenshot shows the Postman application interface. At the top, it says "HTTP localhost:8000/admin.html". Below that, there's a dropdown for "GET" and the URL "localhost:8000/admin.html". Underneath, there are tabs for "Params", "Authorization", "Headers (7)", "Body", "Pre-request Script", "Tests", and "Settings". The "Headers (7)" tab is currently selected, indicated by a red underline. It contains the following table:

Key	Value
Cookie	username=admin
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.34.0
Accept	/*

Below the headers, there are tabs for "Body", "Cookies (1)", "Headers (5)", and "Test Results". The "Body" tab is selected, showing a JSON object:

```
Pretty Raw Preview Visualize HTML
316 // Send an API request to update the stock value
317 console.log('http://127.0.0.1:5000/update_stock/${productName}')
318 fetch('http://127.0.0.1:5000/update_stock/${productName}', {
319   method: 'PUT',
320   headers: {
321     'Content-Type': 'application/json',
322   },
323   body: JSON.stringify({ newStock: parsedStockValue }),
324 }
325 .then(response => {
326   if (response.ok) {
327     stockElement.textContent = `Stock: ${parsedStockValue} units`;
328     location.reload();
329   } else {
330     alert('Failed to update stock. Please try again.');
331   }
332 })
333 
```

Isto pode abrir o site a outras vulnerabilidades mais graves, sendo que o atacante pode agora aceder a todo o código do website facilmente, podendo detetar outras falhas de segurança que pode agora explorar.

Enquanto que o atacante poderia fazer isto sem o cookie, agora tem conhecimento sobre qual é o método de autenticação do website, podendo agora ir ao website normalmente e fazer-se passar por outros utilizadores ou por admin e por exemplo alterar o preço de todos os produtos para 0.

Como explicado na vulnerabilidade acima, na app segura foi implementado um sistema de autenticação com base em JSON Web Tokens.

JSON Web Tokens (JWT) consistem num método de transferência de “declarações” entre sistemas, que no nosso caso são o web server (no qual está o website) e a api.

Estes tokens são criptografados e assinados por uma chave secreta, a qual está definida como variável de ambiente de forma a não ser acedida facilmente.

Cada um destes tokens apresenta uma validade de 30 minutos.

Desta forma, passámos a ter autenticação baseada em server-side.

Cada vez que um utilizador faz login, é criado na api um JWT com a sua role (user ou admin).

Cada vez que um utilizador muda de página, para garantir que está autenticado, é feita uma chamada à api, de modo a validar o seu token.

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://localhost:5000/verify', true);
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.setRequestHeader('Authorization', `Bearer ${token}`);
```

```
@app.route("/verify", methods=["GET"])
@jwt_required()
def verify(): # verify that the user is logged in
    try:
        current_user = get_jwt_identity()
        return jsonify(logged_in_as=current_user), 200
    except Exception as e:
        return Response(status=404, response=json.dumps({"error": "Error while verifying user. Please try later."}))
```

O formato de um JWT pode ser visto na imagem abaixo:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcMVzaCI6ZmFsc2UsImhdCI6MTY5ODk2NTQ1MywianRpIjoiYmQ4MzMMyNjQtY2QxOS00ODZmLTlmZWUtMDR1ZGQ
```

Sendo este token criptográfico e assinado por uma chave secreta, torna-se bem mais difícil para um atacante roubar este token e fazer-se passar por um utilizador do que seria apenas com um cookie como está implementado na app insegura.

Use of Hard Coded Credentials / Missing Encryption of Sensitive Data / Plaintext storage of a Password (CWE-798 / CWE-311 / CWE - 256)

Na app **insegura** as credenciais de acesso a dois utilizadores (dev e admin) apresentam-se expostas sem qualquer encriptação na base de dados.

	username <i>TEXT PRIMARY KEY PRIMARY KEY</i>	pass <i>TEXT NOT NULL</i>
1	admin	admin
2	dev	12345

Isto constitui um grave risco de segurança, visto que um atacante através de uma SQL Injection pode facilmente ter acesso a estes registo e poder logar como administrador no sistema (o utilizador dev não tem nenhum tipo de privilégio), como se pode ver na imagem abaixo:

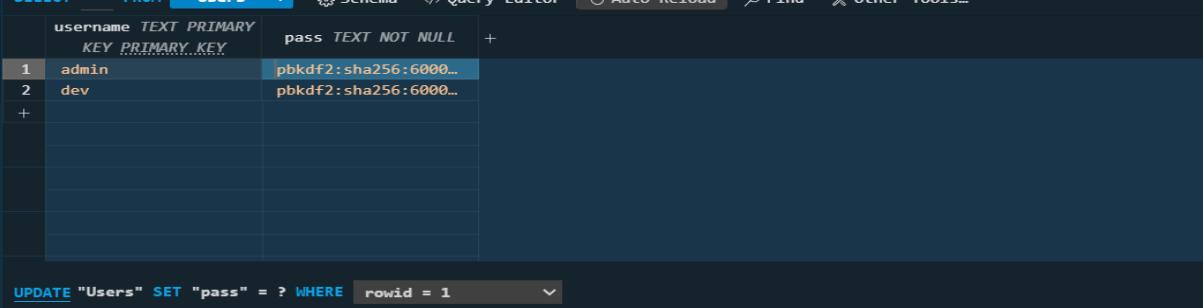
**Looking for:
'UNION SELECT * FROM Users --**

Name	Price
Caneca do Deti	15
Capa pro Telemobil do Deti	25
Chapéu do Deti	10
Pin do Deti	5
Sweat do Deti	25
T-shirt do Deti	20
admin	admin

Através de um SQL Injection básico, o atacante tem agora acesso às credenciais de administrador e pode fazer o que quiser no sistema.

Na app segura, todas as passwords na base de dados são encriptadas com o algoritmo **pbkdf2:sha256**.

Desta forma, mesmo que através de alguma falha de segurança, um atacante consiga aceder aos registos da base de dados, ainda seria bastante difícil descobrir qual é a password (neste caso específico é apenas admin para facilidade de acesso mas assumimos uma password mais segura).



	username	pass
1	admin	pbkdf2:sha256:6000...
2	dev	pbkdf2:sha256:6000...

```
UPDATE "Users" SET "pass" = ? WHERE rowid = 1
1    pbkdf2:sha256:600000$xDuAXVM90hrwU66E$b51633c03f2f8ec0f406f481727c51fbb745fa86cfeaf6479ed0b9249c8404c4
```

Unverified Password Change (CWE-620)

Se um atacante ganhar acesso ao perfil de um utilizador poderá aceder às suas informações mas pior ainda, poderá alterar a sua password ganhando assim acesso fácil à conta e ganhando os seus privilégios. Por isso mesmo para realizar a alteração da password deve ser primeiro pedida a password atual e depois sim dar oportunidade de mudar a mesma.

Na **app insegura** ao clicar no botão de Change apenas pede a nova password:

User: dev

Change password:

New Password

Confirm New Password

Ao clicar no botão Enter vai ser enviado para a Api um dicionário que contém apenas o username e a new_password :

```
function sendNewPass() {
    var newPassword = document.getElementById("new_password").value.trim();
    var confirmPassword = document.getElementById("new_password_confirm").value.trim();

    if (newPassword !== "" && confirmPassword !== "" && newPassword === confirmPassword) {
        var username = document.cookie.split('; ').find(row => row.startsWith('username=')).split('=')[1];
        var data = {
            username: username,
            newPassword: newPassword
        };

        var xhr = new XMLHttpRequest();
        var url = "http://localhost:5000/updatePassword";
        xhr.open("PUT", url, true);
        xhr.setRequestHeader("Content-Type", "application/json");

        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4) {
                if (xhr.status === 200) {
                    alert("Password Updated! Logging out in 5 seconds");
                    var newPasswordInputs = document.getElementById("new_password");
                    var newPasswordConfirmInputs = document.getElementById("new_password_confirm");
                    var enterButton = document.getElementById("enter");

                    enterButton.style.display = "none";
                    newPasswordInputs.style.display = "none";
                    newPasswordConfirmInputs.style.display = "none";
                    var resetText = document.getElementById("resetText").style.display = "block";
                    var resetButton = document.getElementById("reset").style.display = "block";
                } else {
                    alert("Error updating password. Please try again.");
                }
            }
        };
        xhr.send(JSON.stringify(data));
        setTimeout(function () {
            window.location.href = "index.html";
            logout();
        }, 5000);
    }
}
```

Array com o username a new_password:

```
var data = {
    username: username,
    newPassword: newPassword
};
```

Envio dos dados:

```
xhr.send(JSON.stringify(data));
```

Na API é realizada uma query de Update com o user e a new_password sem qualquer verificação:

```
@app.route("/updatePassword", methods=["PUT"])
def updatePassword():
    try:
        data = request.get_json()
        username = data["username"]
        new_password = data["newPassword"]

        conn = sqlite3.connect("LojaDeti.db")
        cursor = conn.cursor()
        print("UPDATE Users SET password = " + new_password + " WHERE username = " + username)
        try:
            cursor.execute("UPDATE Users SET pass = ? WHERE username = ?", (new_password, username))
            conn.commit()
            conn.close()
            return jsonify({"message": "Password updated successfully"})
        except sqlite3.IntegrityError as e:
            print(e)
            return Response(status=409, response=json.dumps({"error": str(e)}))
        except Exception as e:
            print(e)
            return jsonify({"error": str(e)})
    except Exception as e:
        print(e)
        return jsonify({"error": str(e)})
```

Query de update:

```
cursor.execute("UPDATE Users SET pass = ? WHERE username = ?", (new_password, username))
conn.commit()
```

Por sua vez, na **app segura** já aparece outro campo que pede a password atual:

User: dev

Change password:

Change

Actual Password

New Password

Confirm New Password

Enter

Primeiramente verificamos se a nova password tem um formato válido:

```
function sendNewPass() {
    var newPassword = document.getElementById("new_password").value.trim();
    var confirmPassword = document.getElementById("new_password_confirm").value.trim();
    var atualPassword = document.getElementById("atual_password").value.trim();
    if (newPassword == "" || confirmPassword == "" || atualPassword == "") {
        alert("Fill all the fields to change password")
    }
    else if( newPassword.length < 8 || newPassword.search(/[A-Z]/) < 0 || newPassword.search(/\d/) < 0 || newPassword.search(/[@#$%^&*]/) < 0){
        alert(" New Password must be at least 8 characters long, contain a uppercase letter and a special character")
        $("#username").val("");
        $("#password").val("");
        return;
    }
    else if (newPassword == atualPassword) {
        alert("cant change to same password");
    }
    else if (newPassword != confirmPassword) {
        alert("Passwords do not match. Please try again.")
    }
}
```

Agora enviamos para a Api um dicionário mas desta vez já com o campo que é a password atual:

```
var data = {
    atualPassword: atualPassword,
    username: username,
    newPassword: newPassword
};

var xhr = new XMLHttpRequest();
var url = "http://localhost:5000/updatePassword";
xhr.open("PUT", url, true);
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader('Authorization', 'Bearer ' + localStorage.getItem('access_token'));
xhr.onreadystatechange = function () {
    if (xhr.readyState === 4) {
        if (xhr.status === 200) {
            alert("Password Updated! Logging out in 5 seconds");
            var atualPassword = document.getElementById("atual_password");
            var newPasswordInputs = document.getElementById("new_password");
            var newPasswordConfirmInputs = document.getElementById("new_password_confirm");
            var enterButton = document.getElementById("enter");

            enterButton.style.display = "none";
            newPasswordInputs.style.display = "none";
            newPasswordConfirmInputs.style.display = "none";
            atualPassword.style.display = "none";
            var resetText = document.getElementById("resetText").style.display = "block";
            var resetButton = document.getElementById("reset").style.display = "block";
        } else {
            var errorResponse = JSON.parse(xhr.responseText);
            alert(errorResponse.error);
        }
    }
};

xhr.send(JSON.stringify(data));
setTimeout(function () {
    window.location.href = "index.html";
    localStorage.removeItem('access_token');
}, 5000);
```

Na API então realizamos a verificação se a password atual coincide com a password atual inserida e só depois realizamos o Update na base de dados da nova password:

```
atual_password = data["atualPassword"]
conn = sqlite3.connect("LojaDeti.db")
cursor = conn.cursor()
cursor.execute("SELECT pass FROM Users WHERE username = ?", (username,))
password = cursor.fetchone()
if check_password_hash(password[0], atual_password):
    try:
        new_password = generate_password_hash(
            new_password, method="pbkdf2:sha256"
        )
        cursor.execute(
            "UPDATE Users SET pass = ? WHERE username = ?",
            (new_password, username),
        )
        conn.commit()
        conn.close()
        return jsonify({"message": "Password updated successfully"})
    except sqlite3.IntegrityError as e:
        return Response(status=409, response=json.dumps({"error": "Error while updating password. Please try again"}))
    except Exception as e:
        return jsonify({"error": "Error while updating password. Please try again"}),500
else:
    return Response(
        status=404, response=json.dumps({"error": "Wrong actual password"})
    )
```

Weak Password Requirements / Use of Weak Credentials (CWE-521) (CWE-1391)

Nesta seção entramos em detalhes sobre os problemas criados por falta de uma de requisitos de registo fracos e que podem levar a acessos não identificados ou até mesmo violação de dados do site. Aqui no nossa aplicação insegura temos alguns problemas de segurança ao permitir passwords curtas e com poucos caracteres, sem uso de qualquer caracter especial e/ou letra maiúscula e minúscula:

Registros com usernames e passwords fracos podem ser facilmente replicados por atacantes através de ataques de brute force, tendo especialmente em conta que não está implementado um sistema de rate-limiting, permitindo a um ataque fazer inúmeras tentativas até descobrir as credenciais de um utilizador.

```
<script>
    function doRegister() {
        event.preventDefault();
        const username = $("#username").val();
        const password = $("#password").val();

        if (username.length < 4 || password.length < 4) {
            $("#register-success").html(
                "Username and password must be at least 4 characters long"
            );
            return;
        }
    }

```

Já na aplicação segura, para criar uma password existe necessidade de ter uma password com mais de 8 caracteres, conter no mínimo uma letra maiúscula e ter pelo menos uma letra maiúscula:

```
if (username.length < 6){
    alert("Username must be at least 6 characters long")
    $("#username").val("");
    $("#password").val("");
    return;
}

// check if password is atleast 8 characters long, contain a uppercase letter and a special character

if (password.length < 8 || password.search(/[A-Z]/) < 0 || password.search(/[0-9]/) < 0 || password.search(/[!@#$%^&*]/) < 0){

    alert("Password must be at least 8 characters long, contain a uppercase letter and a special character")
    $("#username").val("");
    $("#password").val("");
    return;
}

```

Uma password com estes requisitos torna-se muito mais difícil de descobrir, demorando cerca de 3 anos, segundo a [Hive Systems](#).

Inclusion of Functionality from Untrusted Control Sphere (CWE - 829)

Nesta seção iremos entrar mais em detalhe sobre o CSP inicialmente apresentado na secção de Cross Site Scripting, implementado apenas na **app segura**.

```
class CORSRequestHandler(http.server.SimpleHTTPRequestHandler):
    def end_headers(self):
        self.send_header("Access-Control-Allow-Origin", "self")
        self.send_header("Access-Control-Allow-Methods", "GET, POST, OPTIONS")
        self.send_header("Access-Control-Allow-Headers", "Content-Type" "Authorization")
        self.send_header("Cache-Control", "no-store")
        self.send_header(
            "Content-Security-Policy",
            "default-src 'none';"
            "script-src 'self' cdn.jsdelivr.net code.jquery.com stackpath.bootstrapcdn.com 'nonce-filter-nonce' 'nonce-searchnonce' 'nonce-getcartnonce'"
            "connect-src 'self' http://localhost:5000 ;"
            "img-src 'self' data:;"
            "style-src 'self' cdn.jsdelivr.net 'unsafe-inline' https://fonts.googleapis.com https://fonts.gstatic.com https://cdnjs.cloudflare.com;"
            "font-src 'self' cdn.jsdelivr.net fonts.gstatic.com fonts.googleapis.com cdnjs.cloudflare.com;",
        )
        super().end_headers()
```

Neste CSP estão definidas as seguintes regras:

- Apenas são permitidos scripts do próprio diretório do projeto 'self', sendo os 3 próximos links da fonte utilizada, jquery e bootstrap. Removemos a possibilidade da execução de qualquer script inline por parte do website, de forma a prevenir ataques de XSS, apenas deixando alguns scripts inline correrem, identificados por uma nonce.
- Apenas são permitidas conexões a localhost:5000, que neste caso é o endereço e porto da API, sendo impedidas conexões do site a qualquer outro endereço IP.
- Apenas são permitidas imagens do próprio diretório do projeto.
- São permitidos estilos e fontes da pasta do projeto, inline e a partir das bibliotecas usadas.

No geral tentamos manter este CSP o mais restritivo, principalmente na seção de “script-src”, pelo que tivemos de remover vários scripts inlines que tínhamos na app insegura.

Ao bloquearmos a execução de scripts inline (retirando ‘unsafe-inline’ das script-src), garantimos que mesmo que um payload XSS consiga passar pela primeira barreira de HTML Escaping, vai ser bloqueado pelo CSP e não vai ser executado.

Generation of Error Messages Containing Sensitive Information (CWE - 209)

Mensagens de erro são bastante importantes para um utilizador perceber o que fez de errado.

Também é importante que estas mensagens de erro não revelem informações a mais, pois podem ser exploradas por atacantes mal intencionados. Na app insegura, a api retorna diretamente os erros dados para o frontend:

```
except Exception as e:  
    print(e)  
    return Response(status=404, response=json.dumps({"error": str(e)}))
```

Mesmo que não sejam mostrados diretamente no site, esta resposta pode ser acedida através da aba de Network das Developer Tools.

The screenshot shows a browser window with a registration form on the left and developer tools on the right. The registration form has fields for 'Username' and 'Password', and a 'Login' button. Below the button is a red error message: 'Error trying to register'. On the right, the developer tools Network tab is open, showing a failed request for 'register'. The 'Preview' tab of the Network panel displays the JSON response: { "error": "UNIQUE constraint failed: Users.username" }. This response reveals sensitive information about the database constraint.

Neste caso específico, o atacante fica a saber que este utilizador já existe na base dados, mas poderia obter erros com excertos de código da api também por exemplo.

Na app segura, removemos os erros como valores de retorno em caso de exceções, trocando-os por mensagens esclarecedoras e que não revelam informação confidencial.

```
    return jsonify({"error": "User already exists"}), 409
except sqlite3.IntegrityError as e:
    return Response(status=409, response=json.dumps({"error": "This username already exists. Try using a different one"}))
except Exception as e:
    return jsonify({"error": "Error while registering. Please try again"})
```

Conclusão

Em suma, concluímos que este projeto debruçado principalmente sobre XSS(Cross-site Scripting) e SQL Injections nos ajudou a entender o quanto estas vulnerabilidades apresentam sérias ameaças à integridade de qualquer sistema, principalmente a aplicações web e é por isso que elas representam uma grande porção da roda dos ataques cibernéticos.

Gostamos bastante de trabalhar sobre este tema e ficamos felizes com o resultado final do projeto, achando assim que fizemos um bom trabalho enquanto grupo ao longo das últimas semanas. Para além de melhorar o nosso trabalho em equipa, melhoramos também os nossos conhecimentos sobre a temática da cibersegurança e também sobre todas as CWE's existentes. Aprendemos que educar nos sobre este tipo de temáticas é bastante importante de forma a mitigar cada vez mais este tipo de situações e mantermos uma manutenção regular dos nossos sites de maneira a manter uma boa integridade das nossas aplicações web.

Por fim, este projeto demonstrou um compromisso com a segurança e um desejo de contribuir para um ambiente online mais seguro. Esse esforço é louvável e contribui para a conscientização geral sobre cybersecurity.

Bibliografia

- <https://cwe.mitre.org/data/index.html>
- <https://owasp.org>
- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html
- Conteúdos dos guiões das aulas práticas de SIO
- <https://htmlcodex.com> (template utilizada no website)