Project 1 -Vulnerabilities

Work done by:

- Pedro Monteiro – nº mec 97484
- Renato Dias – nº mec 98380
- Eduardo Fernandes – nº mec 98512
- José Trigo – nº mec 98597

Date: Aveiro, 12th november 2021

Course: Security of Information and Organizations

Professors:

- Professor João Paulo Barraca
- Professor André Zúquete
- Professor Vítor Cunha
- Professor Catarina Silva

universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Index

Introduction

According to Security of Information and Organizations' curricular plan, this report is the result of the execution of the first project, which have as main objective the existence of vulnerabilities in software projects, their exploration and avoidance.

We developed an online shop application which includes some vulnerabilities as the CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'), the CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'), the CWE-20: Improper Input Validation, the CWE-434: Unrestricted Upload of File with Dangerous Type and the CWE-311: Missing Encryption of Sensitive Data.

This practical work aims to determine the risk evaluation but also the potential impact of each vulnerability chosen in our application.

Next, we will discuss the expected topics for this report.

Vulnerabilities

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

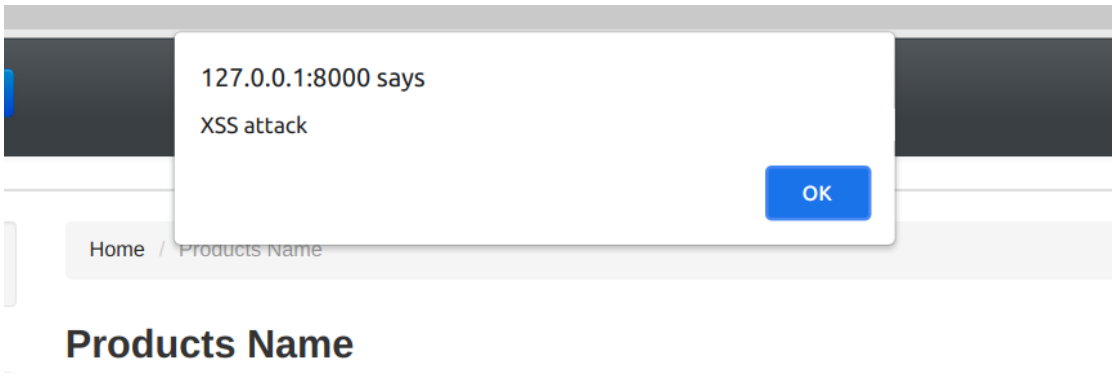
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into benign and trusted websites. XSS attacks happen when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur when a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because of thinking the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

In our application, this vulnerability happens when we introduce for example one of this 3 excerpts of code in the search bar:

- `<script>window.alert("XSS attack");</script>,`
- `<h1>XSS Attack</h1>`
- `<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ4jrXsVu2HuZUeqlkiyLxp4UHZ0sg3GmhN685t0IRke40XZVadvwz6FB9h-__IYHgKxFg&usqp=CAU"</img>`

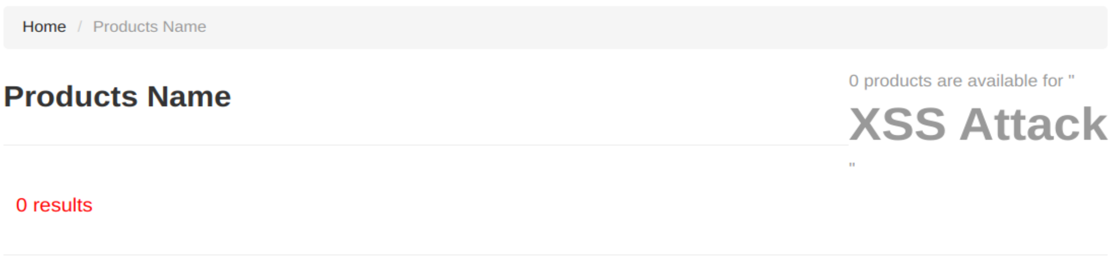
The following images represent the result of the search of each command respectively:



127.0.0.1:8000 says

XSS attack

Result of the search of the first excerpt of code



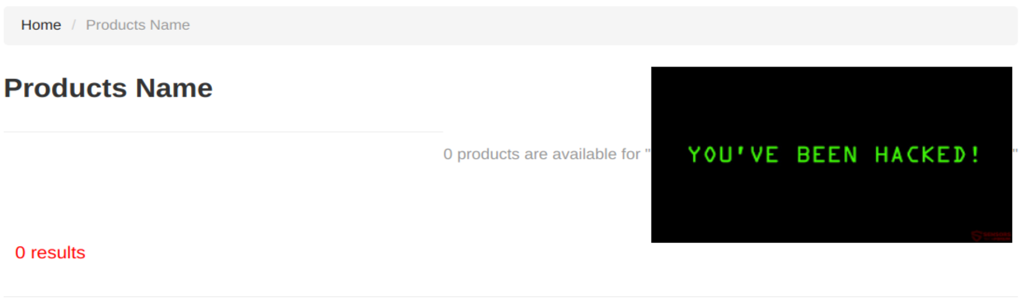
0 products are available for "

XSS Attack

"

0 results

Result of the search of the second excerpt of code



0 products are available for "

YOU'VE BEEN HACKED!

"

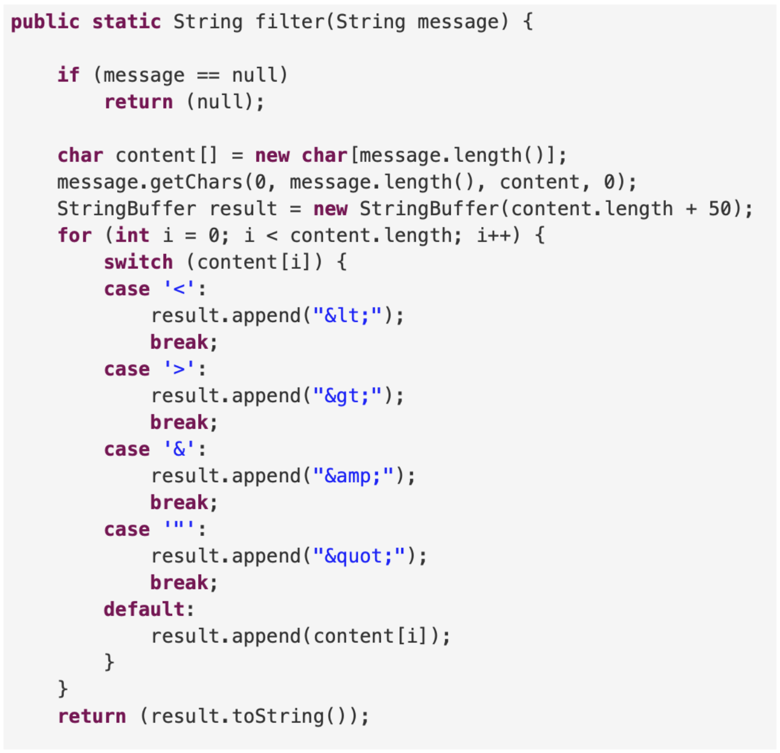
0 results

Result of the search of the third excerpt of code

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

To fix this problem, we used an existing function in *php* that converts the characters of a string to HTML entities. Basically this prevents this type of attack, transforming a potentially dangerous *script* into a normal “non-runnable” string.

Another way to prevent this could be implementing an internal function that verifies if sensitive HTML characters exist and, if so, converts them to friendly ones.



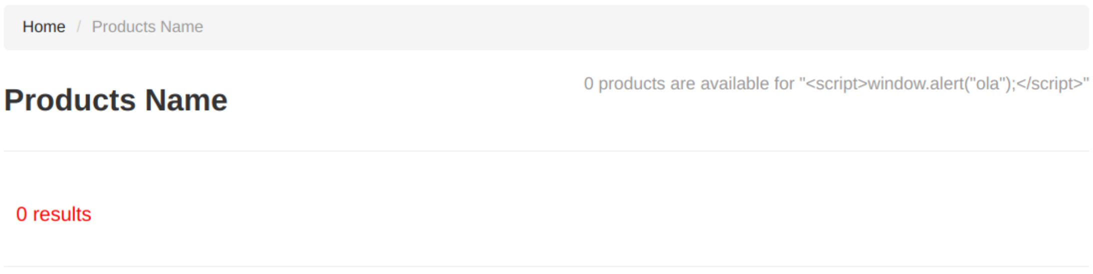
```java
public static String filter(String message) {

    if (message == null)
        return (null);

    char content[] = new char[message.length()];
    message.getChars(0, message.length(), content, 0);
    StringBuffer result = new StringBuffer(content.length + 50);
    for (int i = 0; i < content.length; i++) {
        switch (content[i]) {
        case '<':
            result.append("&lt;");
            break;
        case '>':
            result.append("&gt;");
            break;
        case '&':
            result.append("&amp;");
            break;
        case '"':
            result.append("&quot;");
            break;
        default:
            result.append(content[i]);
        }
    }
    return (result.toString());
```

Example of code in java that converts sensitive HTML characters.

After we implemented this method, Cross-site scripting never worked again.



Home / Products Name

Products Name

0 products are available for "<script>window.alert("ola");</script>"

0 results

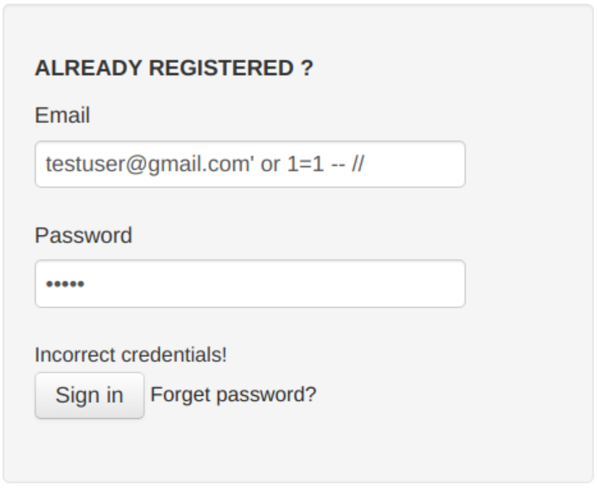
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

More specifically a SQL injection attack consists in a insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.

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

A good practice to avoid this type of attack is to provide the minimum information about the error. The user just needs to know that something has failed, his operation was not successful, there is no need to present the database information.



ALREADY REGISTERED ?

Email: testuser@gmail.com' or 1=1 -- //

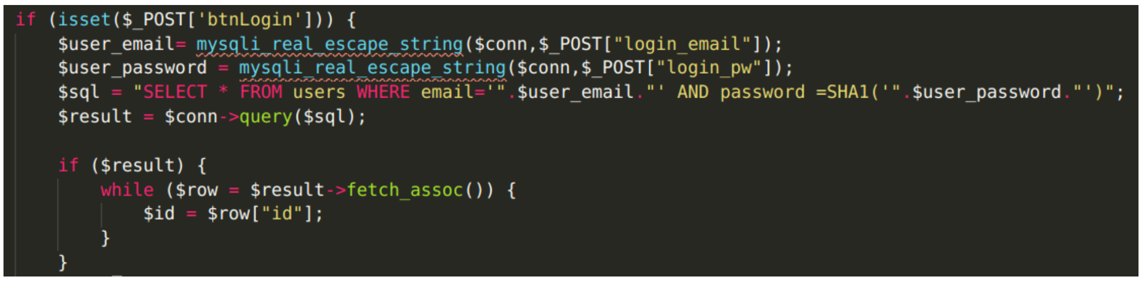
Password: •••••

Incorrect credentials!

Example of a simple error message when trying to inject SQL commands.

Input validation is also a good practice. The validation process is aimed at verifying whether or not the type of input submitted by a user is allowed. Input validation makes sure it is the accepted type, length, format, and so on. Only the value which passes the validation can be processed.

Our solution uses character escape functions for user-supplied input provided by each database management system (DBMS). This is done to ensure that the DBMS never confuses it with the SQL statement provided by the developer. The mysql_real_escape_string() in PHP avoids characters that could lead to an unintended SQL command.



```php
if (isset($_POST['btnLogin'])) {
    $user_email= mysqli_real_escape_string($conn,$_POST["login_email"]);
    $user_password = mysqli_real_escape_string($conn,$_POST["login_pw"]);
    $sql = "SELECT * FROM users WHERE email='".$user_email."' AND password =SHA1('".$user_password."')";
    $result = $conn->query($sql);

    if ($result) {
        while ($row = $result->fetch_assoc()) {
            $id = $row["id"];
        }
    }
```

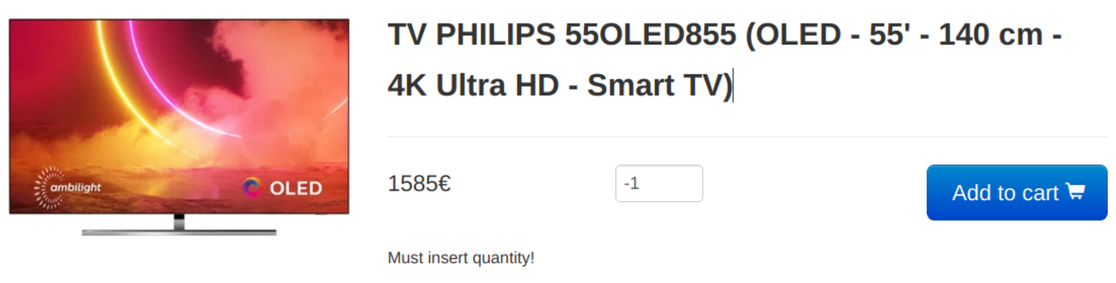
Example of escape functions in php to avoid SQL injection.

CWE-20: Improper Input Validation

The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.

Input validation is a frequently-used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe for processing within the code, or when communicating with other components. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

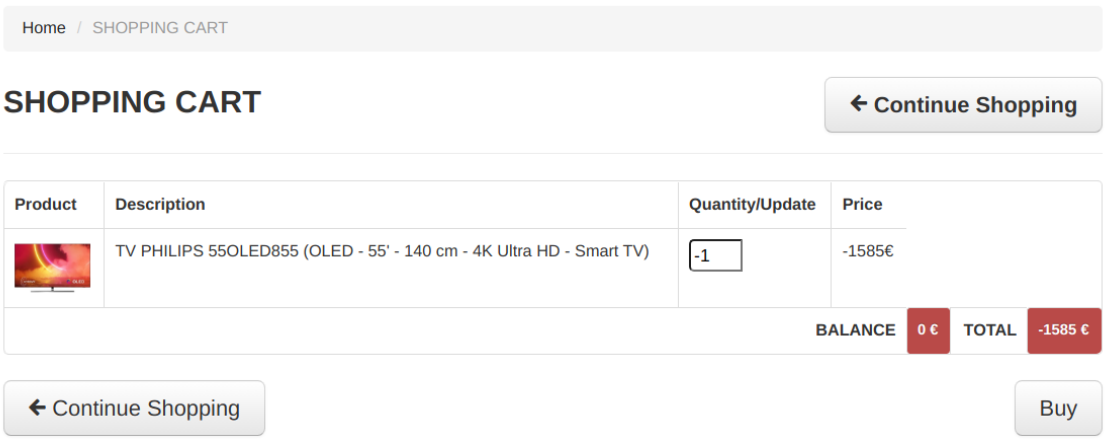
In our case, this vulnerability was affecting the shopping list that the user has. This occurs because in the shopping cart the user can change the value that represents the quantity of a certain product. Assuming that a negative value was entered, the final account balance could also be negative. Therefore, the user is purchasing a product for free and still receiving the value of it. The user can also insert negative values when adding a product to the shopping cart.


TV PHILIPS 55OLED855 (OLED - 55' - 140 cm - 4K Ultra HD - Smart TV)

1585€ -1

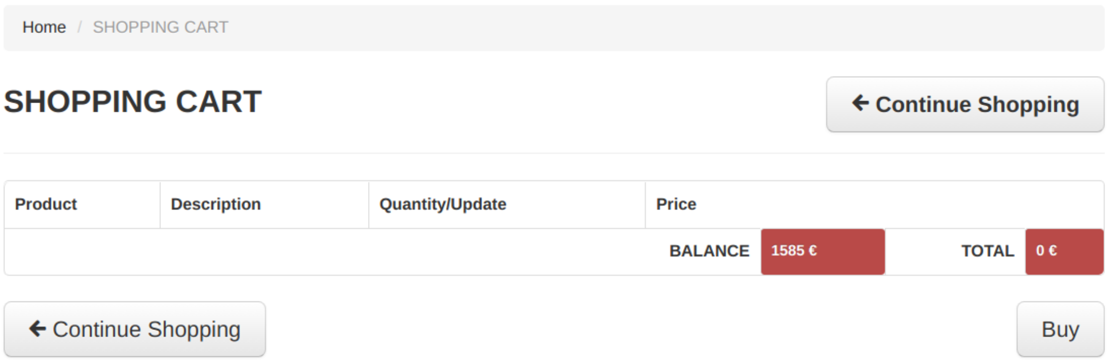
Must insert quantity!

Insert a negative quantity.


Home / SHOPPING CART

SHOPPING CART

Product	Description	Quantity/Update	Price
	TV PHILIPS 55OLED855 (OLED - 55' - 140 cm - 4K Ultra HD - Smart TV)	-1	-1585€
		BALANCE 0 €	TOTAL -1585 €


Home / SHOPPING CART

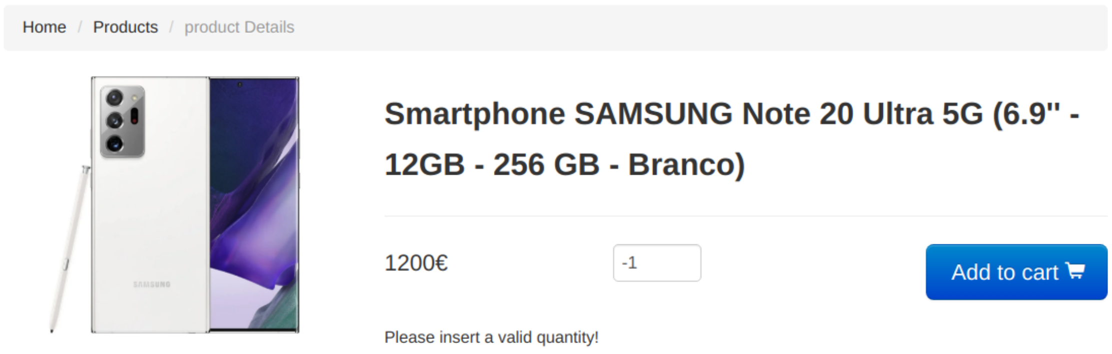
SHOPPING CART

Product	Description	Quantity/Update	Price
		BALANCE 1585 €	TOTAL 0 €

When the user purchases the offer, we can see that his balance increases the value of the product he bought.

CWE-20: Improper Input Validation (Solution)

To solve this problem, the amount entered by the user was verified. If it is below one, an error message is displayed indicating that it is not allowed to enter such values.


Home / Products / product Details

Smartphone SAMSUNG Note 20 Ultra 5G (6.9" - 12GB - 256 GB - Branco)

1200€ -1

Please insert a valid quantity!

As we can see, if the user enters an invalid number then a message is displayed.

CWE-434: Unrestricted Upload of File with Dangerous Type

The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.

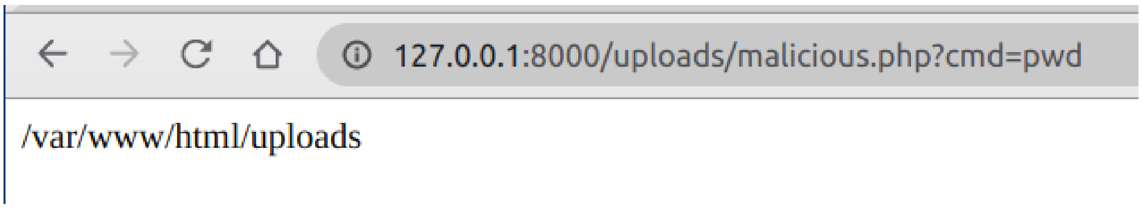
Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

There are many consequences of unrestricted file upload, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

There are two types of problems here. The first is related with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multipart encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. We must validate the metadata extremely carefully before using it.

The other type of problem is related with the file size or content. The range of problems here depends entirely on what the file is used for. To protect against this type of attack, we should analyse everything our application does with files and think carefully about what processing and interpreters are involved.

In our application a *php* file was sent that can be executed by the server. As the uploaded file type is not checked, the *php* file is uploaded to the server. Therefore, a user who tries to break into the system can use the following code `system($_GET['cmd']);` to then execute commands using the URL.



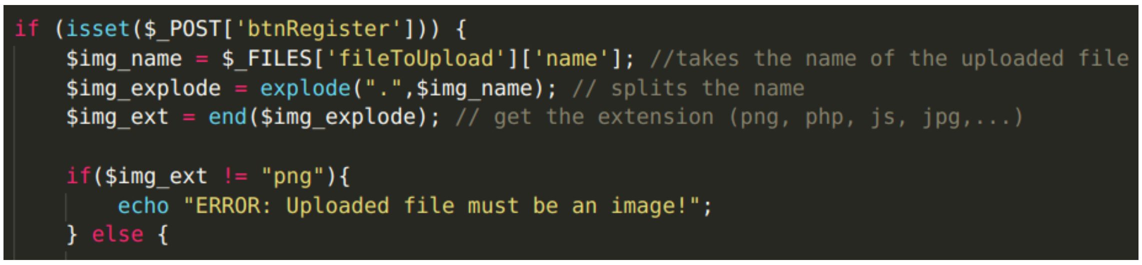
```
127.0.0.1:8000/uploads/malicious.php?cmd=pwd

/var/www/html/uploads
```

At this moment the attacker can run on the URL any command that he wants.

CWE-434: Unrestricted Upload of File with Dangerous Type (Solution)

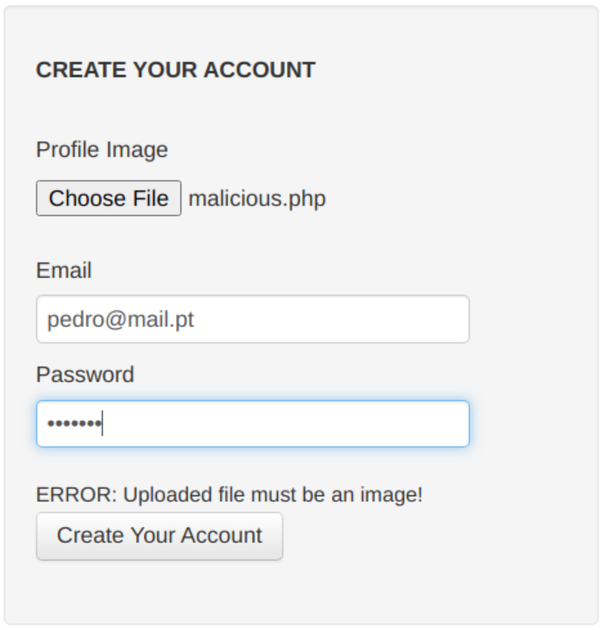
To solve this problem, the type of file that the user tries to submit in the online store is then checked. If the file is of any type other than a *png* file, it is rejected and no longer uploaded to the server. A message then appears to inform the user.



```
if (isset($_POST['btnRegister'])) {
    $img_name = $_FILES['fileToUpload']['name']; //takes the name of the uploaded file
    $img_explode = explode(".",$img_name); // splits the name
    $img_ext = end($img_explode); // get the extension (png, php, js, jpg,...)

    if($img_ext != "png"){
        echo "ERROR: Uploaded file must be an image!";
    } else {
```

Portion of code that verifies the type of the file



CREATE YOUR ACCOUNT

Profile Image

Choose File malicious.php

Email

pedro@mail.pt

Password

•••••••

ERROR: Uploaded file must be an image!

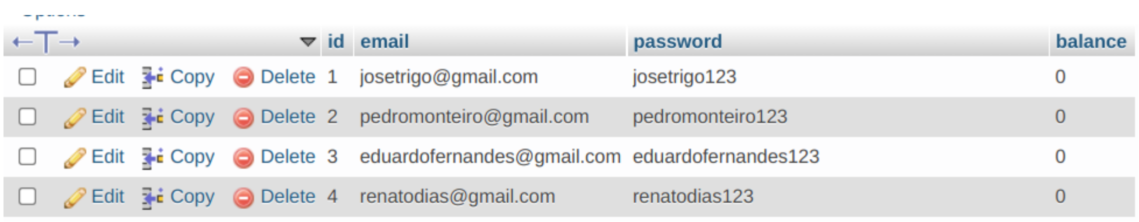
Create Your Account

Error that is presented to user if the file uploaded is not an image

This is a very simple solution, but a necessary one. If this is not done, the attacker could compromise the entire service provided to the user.

CWE-311: Missing Encryption of Sensitive Data

The vulnerability exists due to lack of correct sensitive or important data encryption. Improperly encrypted information can't provide appropriate confidentiality, integrity and accountability of the system. As application doesn't use a secure channel as SSI attackers can easily obtain sensitive data. Vulnerability also allows malicious users to intercept/modify application data.



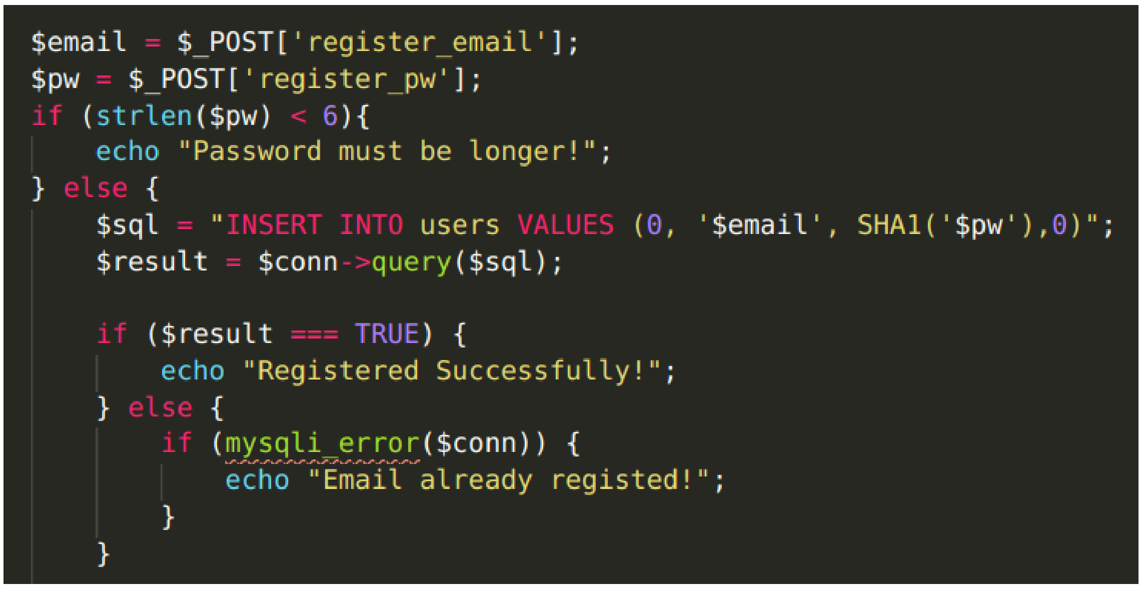
				id	email	password	balance
☐	Edit	Copy	Delete	1	josetrigo@gmail.com	josetrigo123	0
☐	Edit	Copy	Delete	2	pedromonteiro@gmail.com	pedromonteiro123	0
☐	Edit	Copy	Delete	3	eduardofernandes@gmail.com	eduardofernandes123	0
☐	Edit	Copy	Delete	4	renatodias@gmail.com	renatodias123	0

Passwords saved in the database as plain text.

CWE-311: Missing Encryption of Sensitive Data (Solution)

Instead of saving passwords as plain text we used SHA1 and verified if password has at least 6 characters to ensure more security and reliability.

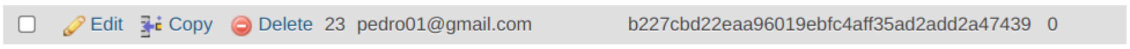
If someone manages to access the database, they will not be able to see at first what data is needed to login, they will have to spend more time to do so, which then makes the attacker's job more difficult.



```php
$email = $_POST['register_email'];
$pw = $_POST['register_pw'];
if (strlen($pw) < 6){
    echo "Password must be longer!";
} else {
    $sql = "INSERT INTO users VALUES (0, '$email', SHA1('$pw'),0)";
    $result = $conn->query($sql);

    if ($result === TRUE) {
        echo "Registered Successfully!";
    } else {
        if (mysqli_error($conn)) {
            echo "Email already registed!";
        }
    }
```

Example of code that helps to protect from CWE-311 Missing Encryption of Sensitive Data



```
☐ Edit Copy Delete 23 pedro01@gmail.com     b227cbd22eaa96019ebfc4aff35ad2add2a47439 0
```

Passwords saved in the database using SHA1 function.