Deti Shop

Project 1 - SIO

Index

Vulnerability	Associated CWEs
1 - Search bar	89
2 - Add product with image	434
3 - Register with weak password	521
4 - User settings URL	488, 522
5 - User settings password change	620
6 - Improperly neutralized product search input	79
7 - Improperly neutralized review critique input	79
8 - Exposed requests	352
9 - Username/Password Disclosure Vulnerability	203
10 - Use of insecure cryptographic algorithm for password	328, 916

1. Introduction

This report documents Project 1 of SIO, in which an online shop specializing in DETI memorabilia at the University of Aveiro was developed with the objective of exploring common software vulnerabilities in a common and typical development scenario.

Two versions of the application were developed: a vulnerable version `app` and a safer version `app_sec`

For each vulnerability found, the related CWEs will be presented, accompanied by a brief explanation.
This is followed with a description of the vulnerability and screenshots/GIFs demonstrating how to replicate it.
Finally, our solution to the problem is described, presenting the changed code, as well as screenshots/GIFs showing the absence of the vulnerability.

2. Overview

To implement this store, we used the following technologies:

1. Backend
   - Flask
   - MySQL database
   - Server-side page rendering with templating using Jinja2
2. Frontend
   - HTML
   - Javascript
   - CSS with the Bootstrap framework
3. Deployment
   - LXD (Linux Containers).

Features implemented:

1. User Management:
   - User registration and login;
   - Update profile info;
   - Password management (change);
   - User roles and permissions (admin, user);
2. Product Catalog:
   - Product listings with details (name, description, price, images);
   - Product categories;
   - Product search functionality;
3. Shopping Cart:

    - Cart management (add, remove);
    - Cart total calculation;
    - Save cart for later;
4. Checkout Process:
    - Shipping information collection;
    - Ability to have a send adress different of Fiscal adress;
    - Order confirmation;
5. User Order History:
    - View and track past orders;
    - View the products ordered, their quantity and price;
6. Inventory and Admin Management:
    - Managing and Tracking product quantities;
    - View Registered Users;
    - View all Orders;
    - Add a new product;
    - Update and remove existing products;
7. Reviews and Ratings:
    - Allow customers to rate and review products;

3. Vulnerabilities

3.1 - Search bar

Weakness

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

    - **Abstract:**

      SQL Injection is a code injection technique that uses a security vulnerability to attack data-driven applications.

      An attacker can exploit this vulnerability in order to bypass authentication and access, modify and delete data stored in the database.

      To perform an SQL injection attack, an attacker must first find vulnerable user inputs within the web page or web application. Then, he can insert a malicious SQL query into the application through the user input.

Description

The home page search bar does not sanitize the user input, so it is possible to perform an SQL injection attack to access the database tables, using queries such as:

```
' UNION SELECT null, username, passwd, null, null, null FROM UserP -- //
```

Demonstration GIF in analysis/gifs folder

However, due to how MySQL cursor.execute() works, it is not possible to perform a second query to the database, meaning we can access and read the database, but not modify it. Here is an example of a query that does not work:

```
'; UPDATE UserP SET perms='A' WHERE username='abcde'; -- //
```

Counteraction

Originally, the search is received like so:

```
cur = mysql.connection.cursor()

cur.execute("SELECT prodID, Product.nome, price, stock, descript, Category.nome AS cat "
            "FROM Product JOIN Category ON (Product.catID = Category.catID) WHERE "
            "Product.nome LIKE '%" + search_name + "%' ORDER BY prodID ASC")
```

To solve this issue, we encapsulated ***search_name*** doing the following **prepared statement**:

```
cur = mysql.connection.cursor()

cur.execute("SELECT * FROM Product WHERE nome LIKE %s ORDER BY nome ASC", (search_name,))
```

This results in the prevention of any type of SQL injection in this input field, as seen below:

GIF CVE 1 Resolved

Demonstration GIF in analysis/gifs folder

3.2 - Add product with image

Weakness

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

    In most sites, users are able to upload files, such as images.

    However, if the site does not check the file type, an attacker can upload a malicious file, such as a PHP file, and then execute it, gaining access to the server.

Description

When a user with admin privileges is logged in, they can add a new product to the store. When adding a new product, the admin can upload an image for the product.

The image is presumed to be a .png or .jpg, but there is no check to see if the file actually matches the extension, meaning any file of any type can be uploaded.

GIF CVE 2

Demonstration GIF in analysis/gifs folder

Counteraction

In *app.py*, the image was received in this way:

```
new_filename = f"{product_id}.{file.filename.split('.')[1]}"
file.save(os.path.join('static/products', new_filename))
```

To counter this, we used a simple guard clause to impede an upload of any other file type that isn't PNG/JPG:

```
if not file.filename.endswith('.png') and not file.filename.endswith('.jpg'):
  flash('Invalid file type, please upload a png or jpg image', 'danger')
  return render_template('add_product.html')
else:
  if file.filename.endswith('.png'):
    new_filename = f"{product_id}.png"
  elif file.filename.endswith('.jpg'):
    new_filename = f"{product_id}.jpg"
  file_path = os.path.join('static/products', new_filename)
  file.save(file_path)
```

So, in *app_sec*, the system is emitting a error message:

GIF CVE 2 Resolved

Demonstration GIF in analysis/gifs folder

3.3 - Register with weak password

Weakness

- **CWE - 521** : Weak Password Requirements
  - **Abstract:**

    Authentication is the process of verifying the identity of a user.

    This is usually done by asking the user to input a username and a password, the password being a secret string that only the user should know.

    However, if the user is able to choose a weak password, like a very short of characters or choose only numbers, it is easier for an attacker to guess it using brute force, and thus gain access to the user's account.

Description

When a user is registering, they can choose their password. However, there are no requirements for the password, meaning the user can choose a weak password, such as "12345678" or "password".

This is also true when a user is changing their password, as there are no requirements for the new password.

GIF CV 3

Demonstration GIF in analysis/gifs folder

Counteraction

In the unsafe application, it was only required that password length would be greater or equal then 8.
Note that for every forms necessary, we used wtforms, that is a Python library that does forms validation and rendering.

```
password = PasswordField('', [validators.length(min=8), validators.DataRequired()],
render_kw={'placeholder': 'Password'})
```

To address this issue, we simply forbidden users from having a password that does not have at least 8 characters, 1 number, 1 UpperCase and 1 LowerCase. We do this check both when SignUp a new user and when the user wants to update his password.

```
password = form.password.data
if len(password) < 8:
    flash('Your password should have at least 8 characters', "danger")
    return render_template("register.html", form=form)
elif not any(char.isdigit() for char in password) :
    flash('Password should have at least one numeral' , "danger")
    return render_template("register.html", form=form)
elif not any(char.isupper() for char in password):
    flash('Password should have at least one uppercase letter', "danger")
    return render_template("register.html", form=form)
elif not any(char.islower() for char in password):
    flash('Password should have at least one lowercase letter', "danger")
    return render_template("register.html", form=form)
```

Demonstration GIF in analysis/gifs folder

3.4 - User settings URL

Weaknesses

- **CWE - 488** : Exposure of Data Element to Wrong Session
  - **Abstract:**

    When a system that has multiple users does not properly isolate data elements, it is possible for one user to access another user's data.

    This can be dangerous, as an attacker can access sensitive information, such as credit card numbers, or even modify the data, such as changing the user's password.
- **CWE - 522** : Insufficiently Protected Credentials
  - **Abstract:**

    There are several instances where sites need to verify the identity of a user, such as when a user is logging in, or when a user is changing their password.

    However, if the site does not protect the credentials, an attacker can intercept them and gain access to the user's account.

Description

In the user settings page, the user can change their personal information. However, there is no verification of the user's identity, meaning you can change another user's information by simply changing the `user` parameter in the URL.

Demonstration GIF in analysis/gifs folder

Counteraction

To solve this problem, we simply added an if statement that compares the `usrID` passed as an argument in URL with the `uid` that is generated by MySQL:

```
if result['usrID'] == session['uid']:
```

Demonstration GIF in analysis/gifs folder

3.5 - User settings password change

Weakness

- **CWE - 620** : Unverified Password Change
  - **Abstract:**

    This vulnerability occurs when a user is able to change their password without having to input the old password.

This can be dangerous, as an attacker can change the password of a user without having to know the old password, essentially locking the user out of their account.

Description

When a user is logged in, they can change their password by going to their profile settings, writing the new password and clicking "Update Settings", without having to input the old password.

Demonstration GIF in analysis/gifs folder

Counteraction

We created a nwe page to update password with a new form, containing 3 fields: `Old Password`, `New Password` and it's confirmation. The application also validates if the `Old Password` given is correct and if the `New Password` is equal to `Confirm Password`

Demonstration GIF in analysis/gifs folder

3.6 - Improperly neutralized product search input

Weakness

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

    User is able to input HTML elements in the product search bar. The server processes this input as effective HTML code, rendering it.

Description

Using the product main page or manipulating the *search* URL parameter used for search queries, a user can input HTML elements which are effectively rendered as part of the search result page in the search term field.

This allows for the injection of dangerous Javascript code and HTML elements in **Reflected XSS** attacks (Non-Persistent).

An example of this attack may involve the sharing of a malicious link with the dangerous code injected in the URL to an oblivious website user. This dangerous code could leverage other vulnerabilities, such as 4 and 5, to do unsolicited operations like password changes. On the other hand, it could also do other vulnerability-independent unsolicited operations, like ordering all shopping cart items.

Demonstration

Here's an example of an attack against a specific individual named Iris Santos. We'll attack her by giving her a malicious link with a script that requests all shopping cart items to be delivered to her:

Here is what the script looks like:

```
<script>
data = {
        first_name : "Iris",
        last_name : "Santos",
        email : "dog@gmail.com",
        mobile : 333222111,
        address : "sample",
};

const XHR = new XMLHttpRequest();
const FD = new FormData();

const urlParams = new URLSearchParams(window.location.search);

const usrID = 2;

for (const [name, value] of Object.entries(data)) {
    FD.append(name, value);
}

let host = location.host;

XHR.open("POST", "http://"+ host +"/shoppingcart?user=" + usrID);

XHR.send(FD);

</script>
```

This is what the malicious hyperlink looks like:

```
http://localhost:5000/search?q="<script> data = { first_name : "Iris", last_name : "Santos", email : "dog@gmail.com", mobile : 333222
```

Here's a demonstration of its effects from Iris Santos' perspective:

Demonstration GIF in analysis/gifs folder

Counteraction

In the vulnerable application, the template used to render search results uses a *safe* pipe, which assumes input can be safely rendered as effective HTML and not simple plain text.

```
<h2 class="my-4">Searched for <b>{{ search_name | safe }}:</b></h2>
```

To fix this, we simply remove the *safe* pipe, rendering injected HTML as plain text and neutralizing its dangerous effects.

3.7 - Improperly neutralized review critique input

Weakness

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

    User is able to input HTML elements in the product search bar. The server processes this input as effective HTML code, rendering it.

Description

Using the review feature on a product's page, a malicious user can input HTML elements in a review's critique section, which will be rendered as effective HTML when other users view the product's page.

This allows for the injection of dangerous Javascript code and HTML elements in **Stored XSS** attacks (Persistent).

Just like the previous vulnerability (3.6), this dangerous code could leverage other vulnerabilities, such as 4 and 5, to do unsolicited operations like password changes. On the other hand, it could also do other vulnerability-independent unsolicited operations, like requesting all shopping cart items.

GIF CVE 7

Demonstration GIF in analysis/gifs folder

Counteraction

In the vulnerable application, the template used to render review critiques uses a *safe* pipe, which assumes input can be safely rendered as effective HTML and not simple plain text.

```
<p><b>Comment: </b>{{ review.critique | safe}}</p>
```

To fix this, we simply remove the *safe* pipe, rendering injected HTML as plain text and neutralizing its dangerous effects.

3.8 - Exposed requests

Weakness

- **CWE - 352** : Cross-Site Request Forgery (CSRF)
    - **Abstract:**

      Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

      CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

Description

When a user is logged in, they can change their personal information, such as their name, email and password. However, there is no CSRF token, meaning an attacker can perform a CSRF attack to change the user's information.

GIF CVE 8

Demonstration GIF in analysis/gifs folder

Demonstrations

To test if our application was vulnerable to CSRF attacks, we developed a simple website with a hidden form that, if the user pressed the button "Click here to win" the hidden form would be submited and sent to the endpoint indicated, in this case it would be sent to the endpoint settings to change the user's settings.

```
 <form hidden id="hack" target="csrf-frame" action="http://localhost:5000/settings?user=1" method="POST" autocomplete="off">

        <input type="text" class="form-control" name="first_name" id="first_name" placeholder="First Name" value="Totally">

        <input type="text" class="form-control" name="last_name" id="last_name" placeholder="Last Name" value="Hacked">

        <input type="text" class="form-control" name="username" id="username" placeholder="Username" value="zzz">

        <input type="text" class="form-control" name="email" id="email" placeholder="Email" value="Sorry Bro">

        <input type="password" class="form-control" name="password" id="password" placeholder="Password" value="you've done it right

        <input type="text" class="form-control" name="description" id="description" placeholder="Description" value="Nice One">

        <input type="text" class="form-control" name="phone_number" id="phone_number" placeholder="Phone Number" value="432785198">

    </form>
    <div class="blur">
        <div class="contain">
            <h1>Win a free DETI T-Shirt</h1>
            <h3 style="margin-bottom: 10px;">Win a free DETI T-Shirt without having to pay for it!</h3>
            <img src="/app_sec/static/products/1.png" style="width: 300px; margin-left: 250px; margin-bottom: 30px;">
            <button onclick="hack();" type="button" id="button" class="text-center btn btn-success"> Click Here to Win!</button>
            <br />

            <div id="warning"></div>
        </div>
    </div>
```

Counteraction

To avoid possible CSRF attacks from outsiders first we changed the app secret key from `app.config.secret_key = "sio_first_project"` to `app.config.secret_key = os.urandom(24)` so that the secret key would change every time a new session has initiate. We also added the `CORS(app)`, cause CORS is a security feature to control web page requests made to a different domain than the one that served the web page, with this addition this would prevent from possible CSRF's attacks. As a final security check we added the following if statements.

```
if 'user' in request.args:
    q = int(request.args['user'])
    if 'uid' in session and q == session['uid']:
    # Rest of the code
```

To check if the user id exists in the URL stored in the value "user" and if that id is the same as the one stored in the session dictionary.

Demonstration GIF in analysis/gifs folder

3.9 - Username/Password Disclosure Vulnerability

Weakness

- **CWE - 203** : Observable Discrepancy
    - **Abstract:**

      The product behaves differently or sends different responses under different circumstances in a way that is observable to an unauthorized actor, which exposes security-relevant information about the state of the product, such as whether a particular operation was successful or not.

Description

When a user tries to login, the application gives different messages for when an incorrect username is supplied, versus when the username is correct but the password is wrong. This difference enables a potential attacker to understand the state of the login function, and obtain half of the necessary authentication credentials.

GIF CVE 9

Demonstration GIF in analysis/gifs folder

Counteraction

To give a more abstract message to the user's, we give the same message for both cases.

```
if result > 0:
            data = cur.fetchone()
            password = data['passwd']
            uid = data['usrID']

            if hashlib.sha256(password_cand.encode()).hexdigest() == password:

                session['logged_in'] = True
                session['uid'] = uid
                session['username'] = username

                if data['perms'] == 'A':
                    session['is_admin_logged_in'] = True
                    return redirect(url_for('admin'))
                else:
                    session['is_admin_logged_in'] = False
                    return redirect(url_for('index'))

            else:
                flash('Incorrect username or password', 'danger')
                return render_template('login.html', form=form)

else:
  flash('Incorrect username or password', 'danger')
  cur.close()
  return render_template('login.html', form=form)
```

GIF CVE 9 Resolved

Demonstration GIF in analysis/gifs folder

3.10 - Use of insecure cryptographic algorithm for password

Weakness

- **CWE - 328** : Use of Weak Hash
    - **Abstract:**

      The product uses an algorithm that produces a digest (output value) that does not meet security expectations for a hash function that allows an adversary to reasonably determine the original input (preimage attack), find another input that can produce the same hash (2nd preimage attack), or find multiple inputs that evaluate to the same hash (birthday attack).

- **CWE - 916** : Use of Password Hash With Insufficient Computational Effort
    - **Abstract:**

The product generates a hash for a password, but it uses a scheme that does not provide a sufficient level of computational effort that would make password cracking attacks infeasible or expensive.

Description

The application was using MD5 hash for password encryption, that has long been considered insecure for cryptographic purposes because it is vulnerable to collision attacks. This means that it is possible to generate different inputs that produce the same MD5 hash, so the password can be discovered.

```
password = hashlib.md5(form.password.data.encode()).hexdigest()
```

Counteraction

To strengthen our password encryption, we changed the Hash Function from MD5() to SHA256()

```
password = hashlib.sha256(password.encode()).hexdigest()
```

4. Conclusion

With this project, we learned a lot about implementing safety measures in web applications and their services, as well as identifying this risks, and how can we mitigate them, using the documentation available like CWE Mitre.

Sources

Images

| DETI Facebook Page (https://www.facebook.com/detiuaveiro/)

Code

| Website Template (https://github.com/mohsinenur/Ecommerce-Website-Using-Python-Flask)

More info about the CWE's found

CWE's Page (https://cwe.mitre.org)