

Project 1 – Vulnerabilities, equipa 18

[CWE-269: Improper Privilege Management 2](#_Toc87479851)

[CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') 4](#_Toc87479852)

[Location: “/” vulnerability in “/registerUser”, target: register user 4](#_Toc87479853)

[Location: “/comments” vulnerability in “/addComent”, target: add comment 6](#_Toc87479854)

[Location: “/” vulnerability in “/comments”, target: preview comment 7](#_Toc87479855)

[Location: “/” vulnerability in “/search”, target: search bar 8](#_Toc87479856)

[Location: “/” vulnerability in “/profile”, target: delete account 9](#_Toc87479857)

[Location: “/” vulnerability in “/profile” and “/search”, target: uname 10](#_Toc87479858)

[Location: “/” vulnerability in “/profile”, target: logout 11](#_Toc87479859)

[CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') 12](#_Toc87479860)

[Location: “/” vulnerability in “/verifyUser” and “/registerUser”, target: login and register user 12](#_Toc87479861)

[CWE-799 Improper Control of Interaction Frequency 14](#_Toc87479862)

# CWE-269: Improper Privilege Management

The software does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor.

With admin privileges the user can delete comments from other users and give them admin permissions as well. This functionality can be seen in the /comments page.

The default admin account is:

* User: admin
* Pass: 123

Regular App:

In this version of the app the attacker can get access to the administrator privileges by:

- finding a way to login to the admin account and then giving the attackers account privileges:

* SQL injection on login page

Graphical user interface, text, application

Description automatically generated

* XSS DOM attack at any page



* XSS DOM attack at /modUser:



Secure App:

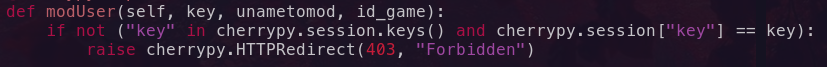
Here the current user is stored in cookies, and the login page is protected against attackers. So it’s much harder to get by trying to force a login into an admin account. Also if the attacker got someone with admin privileges to open a link like this one:



This would not work, since we use the, already used in class, module Fernet, to generate a key that gets saved on the backend and needs to be sent with the form parameters in order to actually work.

Text

Description automatically generated



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

The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

## Location: “/” vulnerability in “/registerUser”, target: register user

Graphical user interface, text, application, chat or text message

Description automatically generated

Regular App:

Text

Description automatically generated

In this version of the app we only check if there’s already a user with the same username in the database before registering, without any restrictions. If the user successfully registers, its name and password get stored in the database.



The username gets displayed on every page of the website after login in. This means that if the user instead of a regular username, inputs a malicious script, this script would run on every page of the website, as long as that user is logged in.

Graphical user interface, text, application, chat or text message

Description automatically generated

In this version of the app, this would be a valid register username. Doing this would give you an alert on every page of the website from now on, when logged in to this account, since the username gets fetched from the database in every page in order to be displayed on top of the site.

Secure App:

Graphical user interface, text, website

Description automatically generated

As explained in SQL Injection - Login/Register Form section, in this version of the app, before doing any type of query to the database, we check for any special characters in the username. The password can have special characters since we only use it for comparisons in the backend. This prevents both Sql injections, and persistent XSS attacks. If the user tries entering any special character, he gets sent back to the registration form.

## Location: “/comments” vulnerability in “/addComent”, target: add comment

Background pattern

Description automatically generated with low confidence

Regular App:

Graphical user interface, application

Description automatically generated

In this version of the app by commenting, the comment is directly written on the database, which is very harmful. In case the comment isn’t plain text, it could be, in fact, a malicious script.



Therefore, if the user’s comment, for example, an alert inside the script tag, that comment is saved on the database and is displayed to every user that enters the comment page for that specific game. This could be worse if the user inputs an external script to steal cookies or other sensitive information about the user.

Secure App:

“swap\_list” is a list of special characters and their respective html token.

Text

Description automatically generated

In the secure version of the app, the username is retrieved from the session that the login was made and the comment special characters are replaced by tokens, using the swap\_sp\_char\_for\_token() function inside sec.py.

With this, the HTML code isn’t read as HTML code but is read as plain text, this replacement makes it possible to handle possible attacks coming from a new comment.



## Location: “/” vulnerability in “/comments”, target: preview comment

Background pattern

Description automatically generated with low confidence

Regular App:

In this version of the app, by previewing a comment of a game we are adding “&preview=comment” to the URL of the page. Because we don’t check the content of the previewed comment it is possible that it contains a malicious script like

“&preview=<script>alert("hello")</script>”. This can be done by adding <script>alert("hello")</script> to the URL or by inserting it into a comment and previewing it.



The indicated link can be sent to someone else and if they access this link, an alert would pop up. This could be threatening as it can also be done with more dangerous scripts.

Secure App:

In the secure version of the app we use the check\_for\_sp\_chars function from “sec.py” to check if the comment has any special characters that could be potentially harmful, if it is harmful send the user back to the comment section of that game with a default preview.

A screenshot of a computer

Description automatically generated with medium confidence

## Location: “/” vulnerability in “/search”, target: search bar

Regular App:

In this version of the app, by searching something on the search bar, we can see that the page URL changes into “search?searchkey=<what\_we\_searched>”. That means we can manually change the link into something like, ”search?searchkey=<script>alert("Flawed")</script>”, or just search for “<script>alert("Flawed")</script>”. This can happen because what is searched in the search bar is displayed beneath as “Search results for <what\_we\_searched>”

Graphical user interface, application

Description automatically generated

We can then send this link to someone and whenever they access the URL, an alert box will pop up with our message. This can, obviously, be further expanded into some more maleficent ends.

Secure App:

In this secure version of the app, before the search is done, the website will check if what you searched for includes special characters, such as “ < ”, “ > ”, “ / “, and more (can be seen in “sec.py” in lines 3 to 12, on the swap\_list), and if so, will swap them in a way that they are no longer harmful to the website by making sure that HTML code isn’t read as HTML code but as plain text, using the “swap\_sp\_char\_for\_token()” function.

A screenshot of a computer

Description automatically generated with medium confidence

## Location: “/” vulnerability in “/profile”, target: delete account

Graphical user interface, text, chat or text message

Description automatically generated

Regular App:

In this version of the app, it’s possible to change profile by manually changing the uname in the URL which in this case is admin “ /profile?uname=admin”. This makes it so that anyone could access every profile.

A screenshot of a computer

Description automatically generated with medium confidence

Secure App:

In this version of the app, as the profile page is created, we generate a random key that is associated with that profile. This makes sure that without this key, it’s not possible to delete the account. We generate this key in the same way we did in class, using the Fernet module and store it in cherrypy.session.

A screenshot of a computer

Description automatically generated with medium confidence

## Location: “/” vulnerability in “/profile” and “/search”, target: uname

Regular App:

In this version of the app, we can observe that the URL is in this format: “search?uname=<NameOfProfile>”.

In this case, we logged in with the ‘andre’ user.

Graphical user interface, application

Description automatically generated

The flaw found in this section of the website is that, by changing the URL into something like “?uname=randomname“, we can overpass the Login/Register mechanism, now being treated as ”randomname” in every part of the website.

Graphical user interface, application

Description automatically generated

The “uname” bit of the URL is also susceptible to the insertion of scripts, the same way as mentioned on the search bar, by changing the section in front of “uname” into something like uname=<script>alert("Flawed")</script>”.

Secure App:

In this version of the app, we can see that the URL no longer has the “?uname=“ parameter, meaning that we can no longer override the login/register mechanism by changing the URL.

This is done by treating the username as a cookie instead of passing it as an argument to the functions.

A screenshot of a computer

Description automatically generated with medium confidence

## Location: “/” vulnerability in “/profile”, target: logout

Regular App:

In the regular app, as shown on the XSS DOM - Uname, in the /profile section of the website, we can simply change the login username by changing the URL. When using that username, you can still Log Out of the website like if you were using the account you logged in with.

Secure App:

Just using cookies wouldn’t be enough to be safe here. If the attacker sent a link like this:



The currently logged in user would get logged out. In order to solve this issue we decided to implement the key test (See the CWE-269 section for more information on this topic).

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

The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

## Location: “/” vulnerability in “/verifyUser” and “/registerUser”, target: login and register user

Graphical user interface, text, application

Description automatically generated

Regular App:

Text

Description automatically generated

Submits form parameters (uname, psw) to “/verifyUser”(login) or “/registerUser”(register), then the following sql query get executed, “SELECT \* FROM User WHERE username=uname AND password=psw;”.

If the user is trying to login, he will successfully do it if anything gets fetched. If he is trying to register, the opposite is applied.

As you can see from the screenshot above, the input from the user doesn’t get any type of filtering and, as long as the executed query returns something, you are able to login.

This means that if you try to login with any valid username followed by “‘ -- //”, only this part gets executed: “SELECT \* FROM User WHERE username=uname” and “AND password=psw;” is commented out.

Graphical user interface, text, application, chat or text message

Description automatically generated

Entering the input above would let you proceed to the search page, since admin is a valid account.

Secure App:

Text

Description automatically generated

On the secure app we do two things to prevent attacks, on the “/” page. If the user is trying to login we check for special characters and swap them out for the equivalent html token. By doing this we protect our database from sql attacks.

Graphical user interface, text, website

Description automatically generated

During the registration process we check for special characters in the and the username field, also we check for users trying to create accounts with no password, sending them back if this verifies. This prevents both sql injections and persistent xss attacks(will be explained with more detail in: XSS Persistent - registerUser())

# CWE-799 Improper Control of Interaction Frequency

The software does not properly limit the number or frequency of interactions that it has with an actor, such as the number of incoming requests.

In the app version, it’s possible to develop a script to spam multiple comments from multiple accounts at the same time. With this, the server and security would be compromised. In the app\_sec version with a cooldown system, we could manage the login, register and add a new comment.

|  |  |  |
| --- | --- | --- |
| COOLDOWN SYSTEM | Maximum strikes | Cooldown time |
| login | 10 | 1 day |
| register | 3 | 1 day |
| addComment | 5 | 15 minutes |

After the third cooldown, within 10 minutes, triggered in the login or register form, the IP address is permanently banned, meaning that is moved to the blacklist and cannot enter the website if he tries, the server raises a forbidden error with the status code 403.

Graphical user interface, text

Description automatically generated

We used the handle\_check\_ip\_request function to control if a request from that IP can be made, in case of permanent ban, and manage all the new IPs

Text

Description automatically generated

For checking if that IP is going to get a permanent ban or for checking if an IP is currently on cooldown.

Text

Description automatically generated

Otherwise, will send to the respective function to check if the number of actions doesn’t surpass the cooldown defined for that action.

Text

Description automatically generated

Also, all the data is written/read from a JSON file.