

eHealthCorp Vulnerabilities

Course: SIO - Security of Information and Organizations

Date: 2022/2023

Students: 102435: Rafael Remígio (<https://github.com/Rafael-Remigio>)
97151: Bruno Rocha Moura (<https://github.com/BrunoRochaDev>)
104360: João Teles Correia (<https://github.com/jteles277>)

Introduction

This assignment will address the presence of vulnerabilities in software projects. For this task, we developed a simple web application for a fictitious health clinic purposely built with several security flaws and weaknesses. Furthermore, a robust version of the application was also developed with proper vulnerability prevention.

Each of the vulnerabilities will be discussed throughout this report, as well as what measures were taken to solve them. To categorize and evaluate them, we used [Common Weakness Enumeration \(CWE\)](#) and [Common Vulnerabilities and Exposures \(CVE\)](#) as guidelines.

The code [repository](#) contains two versions of the web application, one vulnerable to the attacks discussed in this report and the other with safety measures taken to prevent them.

Vulnerabilities Explored

- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
 - Stored XSS
 - Reflected XSS
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- CWE-1336: Improper Neutralization of Special Elements Used in a Template Engine
- CWE-352: Cross-Site Request Forgery (CSRF)

Web Application (Software Architecture)

The web application consisted of a front-end built with HTML, CSS and vanilla JavaScript and a Flask back-end with the module SQLAlchemy for a SQLite persistent database. Login authentication was done with Flask's built-in tools.

The project is containerized using Docker both from a DevOps perspective as well as a safety and reliability perspective.

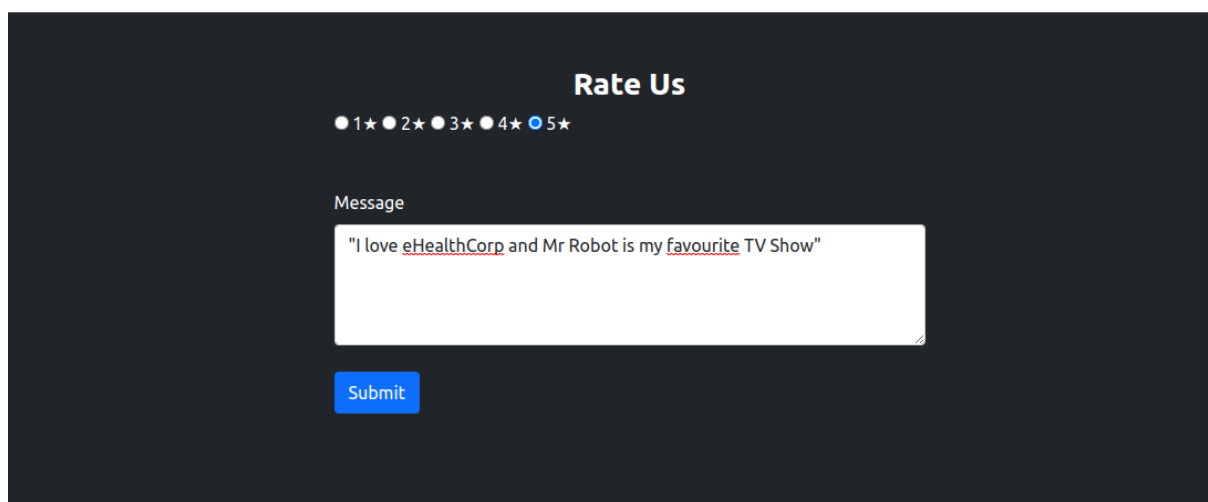
Vulnerabilities

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - Score 7.5

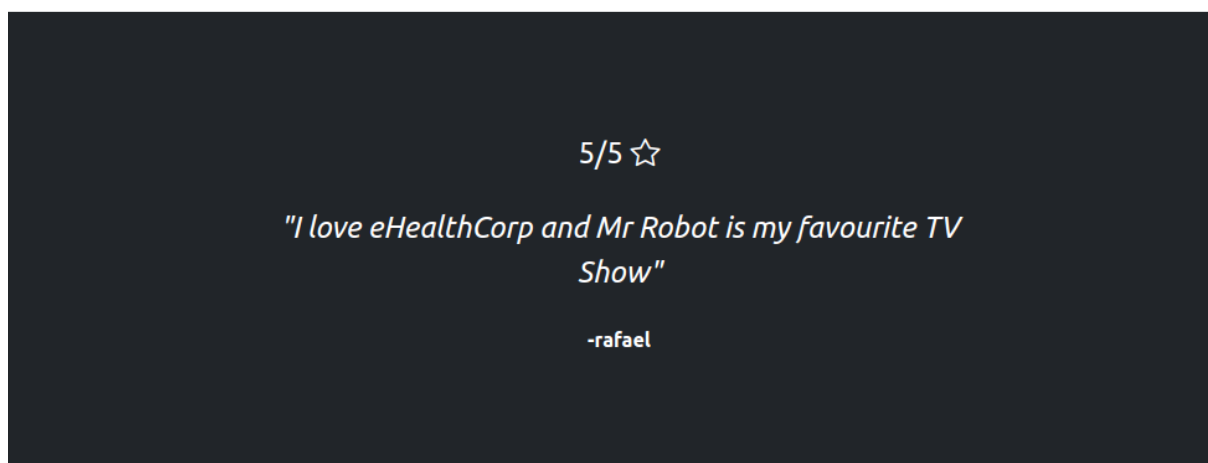
This vulnerability happens because the software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users ([CWE](#)).

The web application allows an user to make a review of eCorp's services, consisting of a rating and a brief text message. User reviews are stored in the database.

There is a dedicated section in the homepage to display the most recent five star rating given by a user.



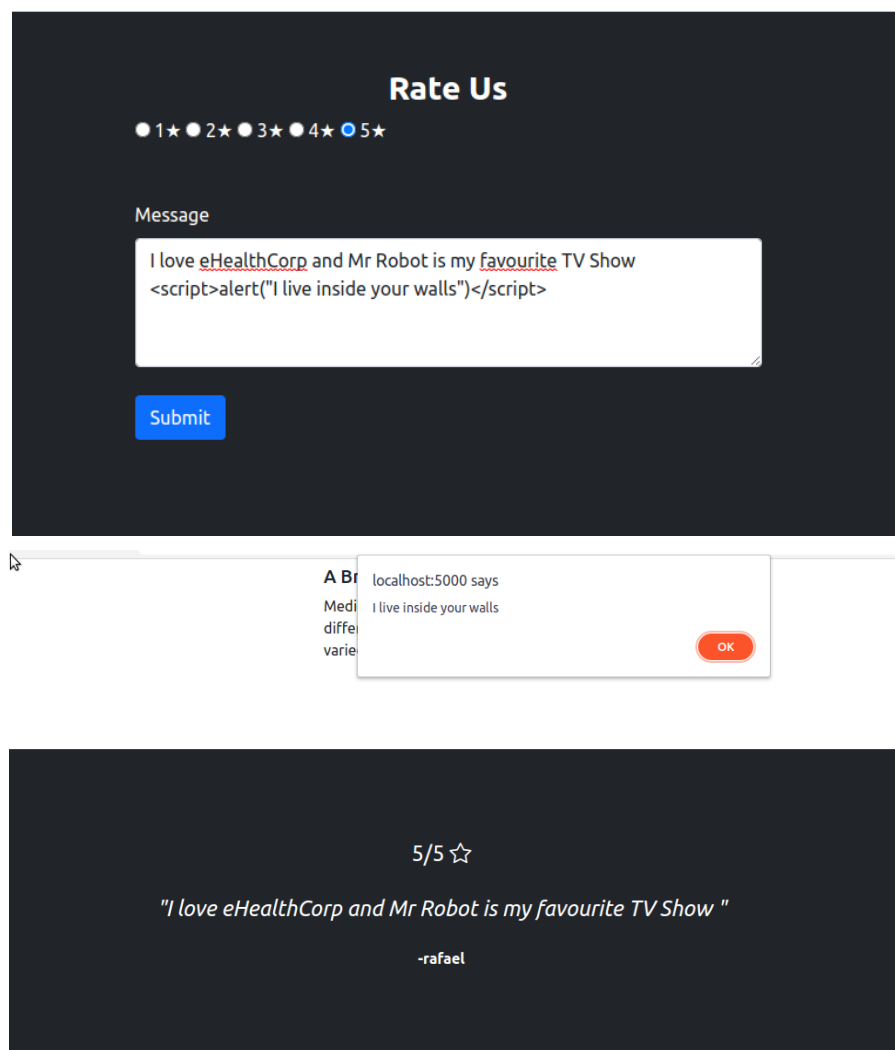
The screenshot shows a dark-themed web interface with a section titled "Rate Us". Below the title is a row of five star icons, with the fifth star (5★) selected and highlighted in blue. Underneath the stars is a label "Message" followed by a white text input field. The input field contains the text: "I love eHealthCorp and Mr Robot is my favourite TV Show". Below the input field is a blue button labeled "Submit".

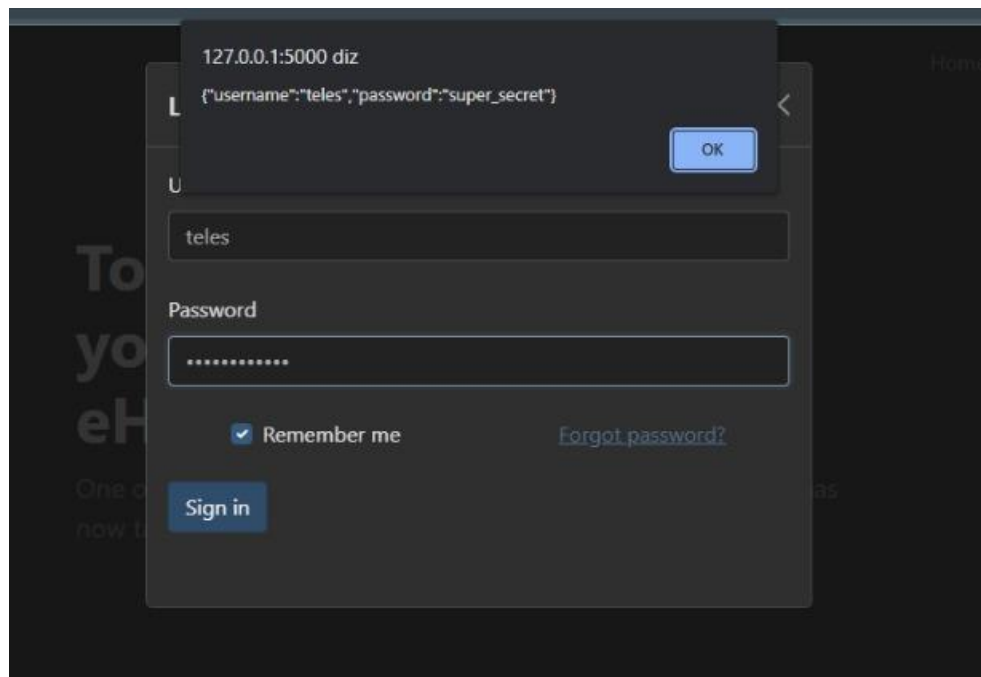
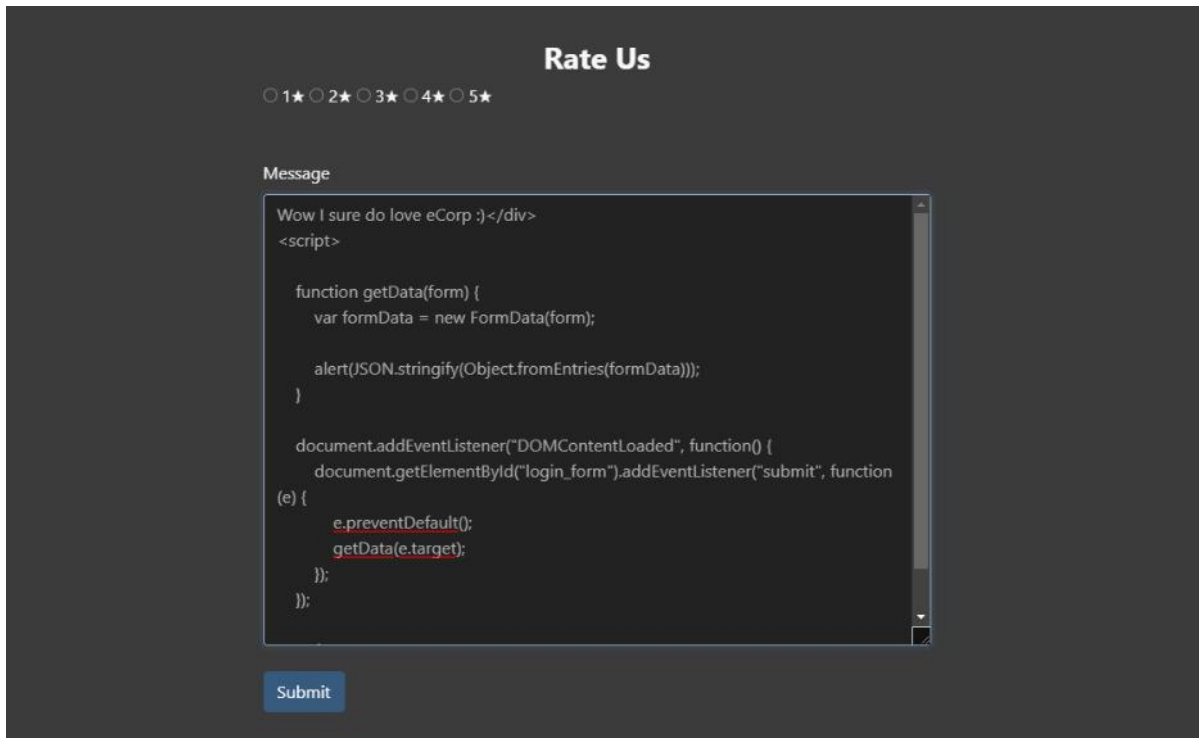


Issue

Since user input is never neutralized, an attacker can inject malicious javascript code utilizing Document Object Model tags. This document will be served to an unsuspecting user and allow the attacker to execute malicious code in the user's browser.

This is both a **Stored-XSS** and a **DOM-Based-XSS**





Vulnerability Scoring and Impact

This vulnerability comprises Confidentiality at a very high level allowing access to a Users personal information and performing actions on their behalf with the use of "Cookies". Cookie Hijacking is very prevalent and gives an attacker full access to your private details.

Base Score		7.5 (High)
Attack Vector (AV) <input checked="" type="radio"/> Network (N) <input type="radio"/> Adjacent (A) <input type="radio"/> Local (L) <input type="radio"/> Physical (P)		
Attack Complexity (AC) <input checked="" type="radio"/> Low (L) <input type="radio"/> High (H)		
Privileges Required (PR) <input checked="" type="radio"/> None (N) <input type="radio"/> Low (L) <input type="radio"/> High (H)		
User Interaction (UI) <input checked="" type="radio"/> None (N) <input type="radio"/> Required (R)		
Scope (S) <input checked="" type="radio"/> Unchanged (U) <input type="radio"/> Changed (C)		
Confidentiality (C) <input type="radio"/> None (N) <input type="radio"/> Low (L) <input checked="" type="radio"/> High (H)		
Integrity (I) <input checked="" type="radio"/> None (N) <input type="radio"/> Low (L) <input type="radio"/> High (H)		
Availability (A) <input checked="" type="radio"/> None (N) <input type="radio"/> Low (L) <input type="radio"/> High (H)		

Prevention

Generally speaking, a common way to prevent cross-site scripting is by sanitizing the user input. That is to say, disallowing certain characters that are not relevant for the purpose of the input but can allow for injection of html tags directly into the DOM.

More specifically, Flask (which uses Jinja2 as template engine), has HTML escaping enabled by default, making it impossible for raw HTML to be rendered in a template. So securing the application against this kind of attack was as simple as opting back in.

```
<div class="fs-4 mb-4 fst-italic text-white"> "{{ review_content|safe }}" </div>
```

The safe attribute opts out of HTML escaping

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - Score 7.5

This occurs when 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 ([CWE](#))

In the login section the user is asked to provide a Username and a Password. These will be used in the SQLite Query to validate the User's Authentication.

The image shows a web application interface. At the top, there is a navigation bar with links for 'Home' and 'Medics'. Below this, a 'Login' modal is displayed. The modal contains a 'Username' field with the text 'rafael' and a 'Password' field with masked characters. There is a 'Remember me' checkbox that is checked, a link for 'Forgot password?', and a 'Sign in' button. At the bottom of the page, there is another navigation bar showing 'Home', 'Medics', 'Logged in as rafaël', and a 'Logout' button.

Issue

Since the user input is not properly neutralized an attacker can use a SQL Injection to bypass Authentication and even gain control over another user's account.

```
db.engine.execute(f' SELECT username FROM users WHERE  
username="{username}" and password="{password}" ')
```

This query allows any user to gain control over another user's account by simply knowing their Username

The image shows a web application interface. At the top, there is a navigation bar with links for 'Home', 'Medics', and 'Login'. Below this, a 'Login' modal is displayed. The modal has a title 'Login' and a close button. It contains two input fields: 'Username' and 'Password'. The 'Username' field is filled with the text 'rafael\" or 1=1 --'. Below the 'Username' field is a 'Password' field. Under the 'Password' field, there is a 'Remember me' checkbox which is checked, and a link that says 'Forgot password?'. At the bottom of the modal is a blue 'Sign in' button. In the background, below the login modal, there is another navigation bar that says 'Home', 'Medics', 'Logged in as rafa'el', and a blue 'Logout' button.

Another example of this same issue can be found when inserting a contact to a database.

```
SQLcommand = f'insert into contactMessage(textMessage,name,email)
values ("{message}", "{textMessageName}", "{textMessageEmail}")'
```

This insert for example will proceed to drop and delete all records of reviews.

The image shows a 'Contact Us' form. At the top, there is a title 'Contact Us'. Below the title, there is contact information: 'Location: New York City', 'ehealthcorp@among.us', and '222 420 123'. The main section is titled 'Send us a message'. It contains three input fields: a text area for the message (containing the text 'im gonna hack you'), an 'Email' field (containing the text 'asdasd '); drop table reviews --'), and a 'Your Name' field (containing the text 'my name is none of your business'). At the bottom of the form is a blue 'Submit' button.

Vulnerability Scoring and Impact

This vulnerability comprises Confidentiality at a very high level allowing access to a Users personal information and performing actions on their behalf. It also allows deleting and retrieving all data in the database, compromising Integrity at a very high level

Base Score		9.1 (Critical)
Attack Vector (AV)		Scope (S)
<input checked="" type="radio"/> Network (N) <input type="radio"/> Adjacent (A) <input type="radio"/> Local (L) <input type="radio"/> Physical (P)		<input checked="" type="radio"/> Unchanged (U) <input type="radio"/> Changed (C)
Attack Complexity (AC)		Confidentiality (C)
<input checked="" type="radio"/> Low (L) <input type="radio"/> High (H)		<input type="radio"/> None (N) <input type="radio"/> Low (L) <input checked="" type="radio"/> High (H)
Privileges Required (PR)		Integrity (I)
<input checked="" type="radio"/> None (N) <input type="radio"/> Low (L) <input type="radio"/> High (H)		<input type="radio"/> None (N) <input type="radio"/> Low (L) <input checked="" type="radio"/> High (H)
User Interaction (UI)		Availability (A)
<input checked="" type="radio"/> None (N) <input type="radio"/> Required (R)		<input checked="" type="radio"/> None (N) <input type="radio"/> Low (L) <input type="radio"/> High (H)

Prevention

Just as with cross-site scripting, this kind of vulnerability may be resolved by limiting the characters admissible in the input field. Characters such as commas, quotation marks and mathematical operands are the usual culprits.

On our application specifically, we solved the issue by using the proper methods SQLAlchemy provides for writes and reads instead of executing raw SQL directly.

```
user = User.query.filter_by(username = username).first() # Gets the user from a username

if user != None and bcrypt.check_password_hash(user.password, password): # Checks if the password matches
    login_user(user)
```

The proper way to authenticate user login using Flask methods

```
result = db.engine.execute(f'SELECT username FROM users WHERE username="{username}" and password="{password}" ')
records = [r for r in result.fetchall()]

# If there's an user that matches username and password...
if len(records) > 0:
    username = records[0][0]
    login_user(User(username, password))
```

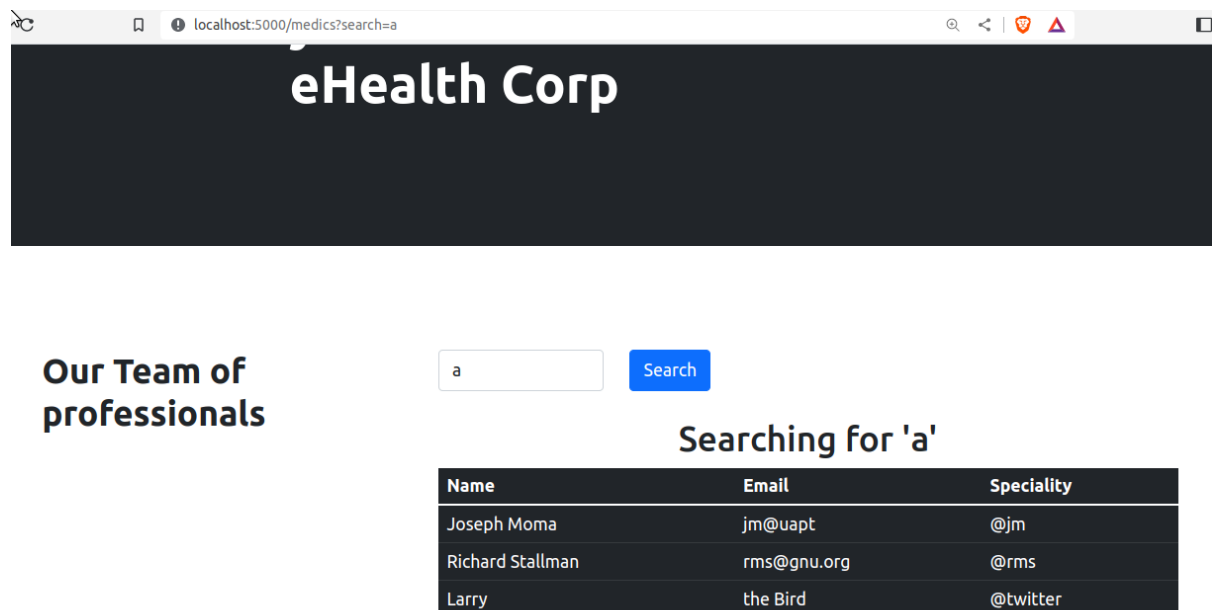
Authentication using raw SQL, subject to injections

CWE-1336: Improper Neutralization of Special Elements Used in a Template Engine - Score 10

This occurs when the WebApp uses a template engine to insert or process externally-influenced input, but it does not neutralize or incorrectly neutralizes special elements or syntax that can be interpreted as template expressions or other code directives when processed by the engine.

In this case the template engine is Jinja used to render HTML pages. If the inputs are not neutralized the application is vulnerable to Template Injection.

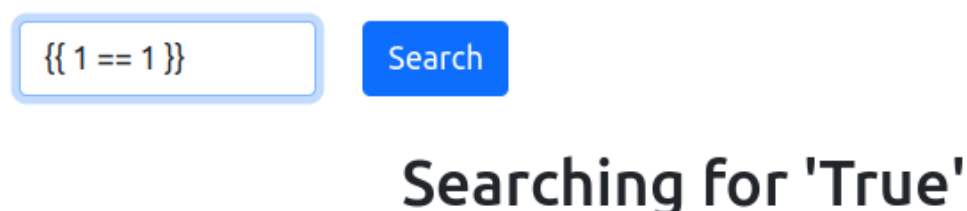
In our application the template engine is used for performing search queries.



Name	Email	Speciality
Joseph Moma	jm@uapt	@jm
Richard Stallman	rms@gnu.org	@rms
Larry	the Bird	@twitter

Issue

Since the inputs are not properly sanitized and neutralized we can perform Server Side Template Injections. SSTI allows executing code at the server's end.



Searching for 'True'

For example, an attacker can execute a query that executes and returns a listing of the files in a directory.

Our Team of professionals

Searching for 'Dockerfile README.md app instance main.py requirements.txt '

Name	Email	Speciality
Joseph Moma	jm@uapt	@jm
Richard Stallman	rms@gnu.org	@rms
Larry	the Bird	@twitter

Vulnerability Scoring and Impact

This would allow crashing a server and having complete access over all records in the database.

Base Score

10.0
(Critical)

Attack Vector (AV)

Network (N)

Adjacent (A)

Local (L)

Physical (P)

Attack Complexity (AC)

Low (L)

High (H)

Privileges Required (PR)

None (N)

Low (L)

High (H)

User Interaction (UI)

None (N)

Required (R)

Scope (S)

Unchanged (U)

Changed (C)

Confidentiality (C)

None (N)

Low (L)

High (H)

Integrity (I)

None (N)

Low (L)

High (H)

Availability (A)

None (N)

Low (L)

High (H)

Prevention

Preventing this vulnerability is as simple as using the Flask method `render_template` instead of `render_template_string`. This way, user input is properly sanitized and isn't treated as python code.

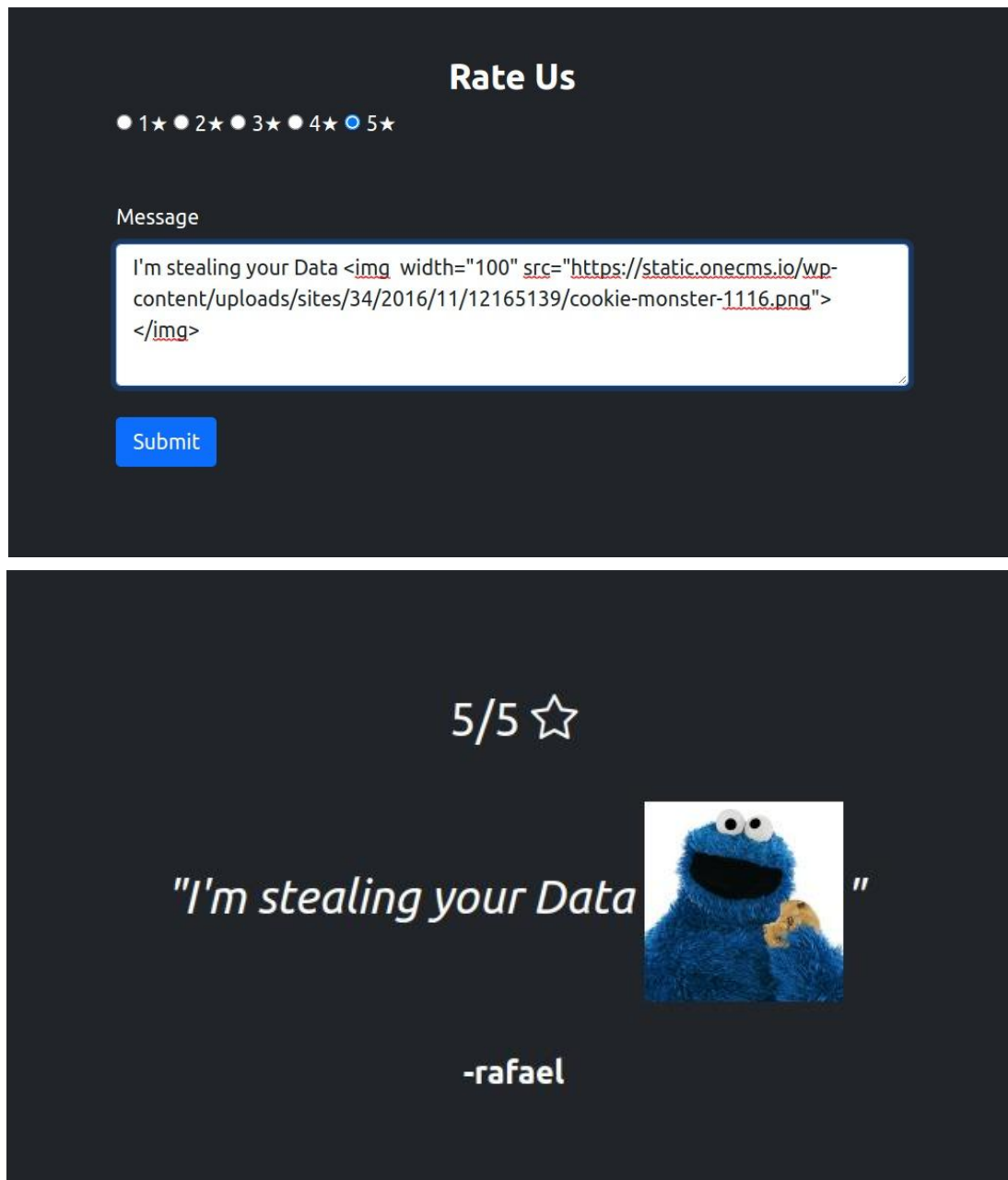
```
return render_template('medics.html', search_term = search_term)
```

```
return render_template_string(medic_html.replace('{search_term}', search_term))
```

CWE-352: Cross-Site Request Forgery (CSRF) - Score 7.5

Because the WebApplication allows Cross-Site-Scripting we can perform requests to other applications on the user's end. If another application/web site does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

Example of testing if CSRF is possible:



Rate Us

● 1★ ● 2★ ● 3★ ● 4★ ● 5★


Message

I'm stealing your Data

Submit

5/5 ☆

"I'm stealing your Data"



-rafael

Issue

An attacker may try to complete a transaction on your behalf using your own cookies. This exploits can be disguised as links or as images for example:

```
<a href="http://bank.com/transfer.do?acct=Bruno&amount=123456">View  
my Pictures!</a>  

```

Vulnerability Scoring and Impact

This vulnerability is closely related with [CrossSite Scripting](#) and a child of [Insufficient Verification of Data Authenticity](#) and although not specific to our WebApplication, can put our users at risk.

Base Score		7.5 (High)
Attack Vector (AV)	Scope (S)	
<input checked="" type="radio"/> Network (N) <input type="radio"/> Adjacent (A) <input type="radio"/> Local (L) <input type="radio"/> Physical (P)	<input checked="" type="radio"/> Unchanged (U) <input type="radio"/> Changed (C)	
Attack Complexity (AC)	Confidentiality (C)	
<input checked="" type="radio"/> Low (L) <input type="radio"/> High (H)	<input type="radio"/> None (N) <input type="radio"/> Low (L) <input checked="" type="radio"/> High (H)	
Privileges Required (PR)	Integrity (I)	
<input checked="" type="radio"/> None (N) <input type="radio"/> Low (L) <input type="radio"/> High (H)	<input checked="" type="radio"/> None (N) <input type="radio"/> Low (L) <input type="radio"/> High (H)	
User Interaction (UI)	Availability (A)	
<input checked="" type="radio"/> None (N) <input type="radio"/> Required (R)	<input checked="" type="radio"/> None (N) <input type="radio"/> Low (L) <input type="radio"/> High (H)	

Prevention

The prevention is the same as the prevention as Cross-Site-Scripting

Conclusion

This assignment gave us a better understanding of security vulnerabilities in software development. Developing both a secure and an insecure application and exploring and exposing vulnerabilities in a WebApplication allowed us to understand these challenges in many different perspectives.