Project 1 (Report)

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

Introduction

Within the scope of the first group project of Security of Information and Organizations, we developed a simple web application for a health clinic that allows an average user to see information about the clinic's doctors and departments, make appointments with a specific doctor, contact the clinic and download personal test results by providing a code. An administrator has the ability to see every appointment and contact that has been made in the clinic.

Both versions of the web application (secure and insecure) were written in Flask using Python 3.8, and MySQL was used for data persistence. The front-end was based on a template developed by Themefisher called Novena[1], that needed some modifications to work in Flask. The project is configured to be run with Docker Compose for ease of use.

- **exams/**
  contains the user's exam results.
- **static/**
  contains all CSS/SCSS files, images, JavaScript libraries necessary to run the application.
- **templates/**
  contains all HTML pages of the application, that use the Jinja templating mechanism to render the information.
- **Dockerfile**
  contains the instructions to assemble a Docker image for the web application.
- **README.md**
  contains a brief overview of relevant information about the web application.
- **app.py**
  contains the business logic of the application. Flask views route requests, interact with the database and render the HTML pages.
- **classes.py**
  contains a wrapper for user entities, that allows the web application to easily store sessions in the browser, keeping the user logged in between pages.
- **db.py**
  contains functions to reset and interact with the database.
- **docker-compose.yml**
  contains the instructions for Docker to setup the containers for the web application and the database.
- **schema.sql**
  contains the SQL instructions that allow the database to be reset to its original state.
- **utils.py**
  contains a function to determine available time slots for appointments, based on the doctors' schedules and already occupied slots.

[1] https://themewagon.com/themes/free-bootstrap-4-html-5-healthcare-website-template-novena/

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

CVSS

- **Score: 9.8**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

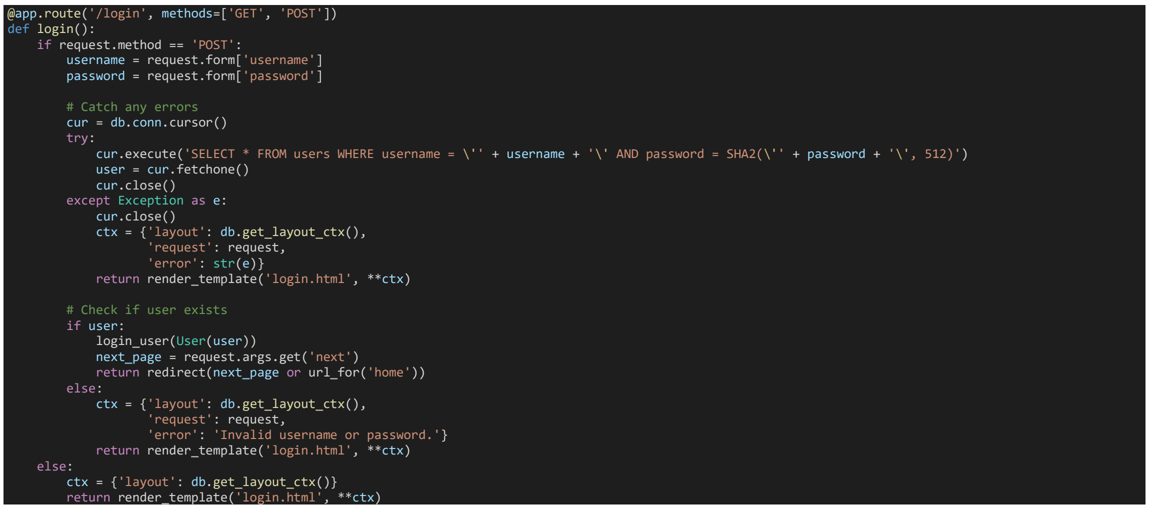
Description

CWE-89 is known for being a basic form of SQL injection, describes the direct insertion of attacker-controlled data into variables that are used to construct SQL commands. This vulnerability occurs when we have unsanitized submission forms.

Potential Impact

An attacker can view, add, delete, or modify information stored in the database with the privileges of the current database user. He can also extract information from the database. In case of a web application, this weakness often leads to a website deface or a database exfiltration.

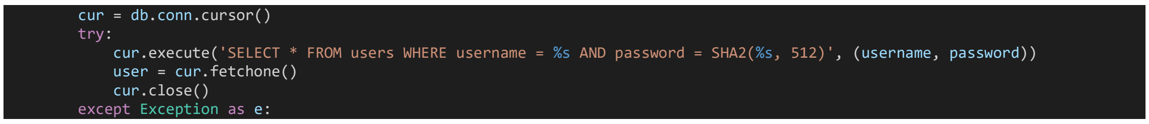
In the insecure application, we can exploit this vulnerability in every page that has a form, in particular the login page. The parameters username and password are concatenated on the query's string, allowing for unsanitized malicious input. When accessed, our login page calls the following function:



```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Catch any errors
        cur = db.conn.cursor()
        try:
            cur.execute('SELECT * FROM users WHERE username = \'' + username + '\' AND password = SHA2(\'' + password + '\', 512)')
            user = cur.fetchone()
            cur.close()
        except Exception as e:
            cur.close()
            ctx = {'layout': db.get_layout_ctx(),
                   'request': request,
                   'error': str(e)}
            return render_template('login.html', **ctx)

        # Check if user exists
        if user:
            login_user(User(user))
            next_page = request.args.get('next')
            return redirect(next_page or url_for('home'))
        else:
            ctx = {'layout': db.get_layout_ctx(),
                   'request': request,
                   'error': 'Invalid username or password.'}
            return render_template('login.html', **ctx)
    else:
        ctx = {'layout': db.get_layout_ctx()}
        return render_template('login.html', **ctx)
```

Correction

In the secure application, instead of concatenating the parameters **username** and **password**, they are parameterized. This prevents any unsanitized input to have an effect on the behaviour of the application.



```python
        cur = db.conn.cursor()
        try:
            cur.execute('SELECT * FROM users WHERE username = %s AND password = SHA2(%s, 512)', (username, password))
            user = cur.fetchone()
            cur.close()
        except Exception as e:
```

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

CVSS

- **Score: 7.2**
- **Vector String:** CVSS:3.1/AV:N/.AC:L/.PR:N/.UI:N/.S:C/.C:L/.I:L/.A:N

Description

The weakness occurs when the application does not perform or incorrectly performs a neutralization of an input data, before displaying it in user's browser. As a result, an attacker can inject and execute arbitrary HTML and script code in user's browser. It has 2 main types:

- **Reflected XSS:** This type describes an error when application reads input data from the HTTP request and reflects it back in HTTP response. The malicious content is never stored in the application and can be viewed only when user follows a specially crafted link.
- **Stored XSS:** This type describes an error when application reads input data from the HTTP request and stores it in database, logs, cached pages, etc. Malicious code can be later executed in user's browser when user visits a vulnerable page.

Potential Impact

This vulnerability allows the attacker to force the server to execute scripts. After successful attack a malicious user can perform a variety of actions: steal user's cookies, modify webpage contents, perform operations with the site within the user’s session. This type of attack is extremely versatile because we can put anything inside the script.

In the insecure application, we need to specifically tell the Jinja templating mechanism that the value of the parameters we are passing on to it are safe! Inside the administrator page template we can find the following content:

```
<tbody>
    {% for c in contacts %}
    <tr>
        <td>{{ c.u_first_name }} {{ c.u_last_name }}</td>
        <td>{{ c.subject|safe }}</td>
        <td>{{ c.message|safe }}</td>
    </tr>
    {% endfor %}
</tbody>
```

Correction

In the secure application, we just needed to omit this instruction, because Jinja sanitizes every data that it renders by default.

```
<tbody>
    {% for c in contacts %}
    <tr>
        <td>{{ c.u_first_name }} {{ c.u_last_name }}</td>
        <td>{{ c.subject }}</td>
        <td>{{ c.message }}</td>
    </tr>
    {% endfor %}
</tbody>
```

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

CVSS

- **Score: 4.3**
- **Vector String:** CVSS:3.1/AV:N/.AC:L/.PR:N/.UI:R/.S:U/.C:N/.I:L/.A:N

Description

When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request.

Taking advantage of the cookies stored in the browser's cache, which are generated to keep the user's session started on a given website for some time, we can use them to make requests on another system using the victim's identity.

Potential Impact

Almost every web application contains functionality that requires certain privileges. If a validation of the request origin is not performed, an unauthenticated user might be able to use this functionality. Depending on the application functionalities, an attacker might be able to gain access to otherwise restricted areas and perform simple tasks such as publishing comments under user's name, creating administrative accounts or use this weakness as a surface for further attacks.

In the insecure application, we take advantage of the previous vulnerability (XSS) to include scripts with dynamic ajax calls, that send the administrator's cookies stored in his browser to an external server.

Correction

In the secure application, by preventing XSS we also prevent CSRF.

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

CVSS

- **Score: 7.5**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

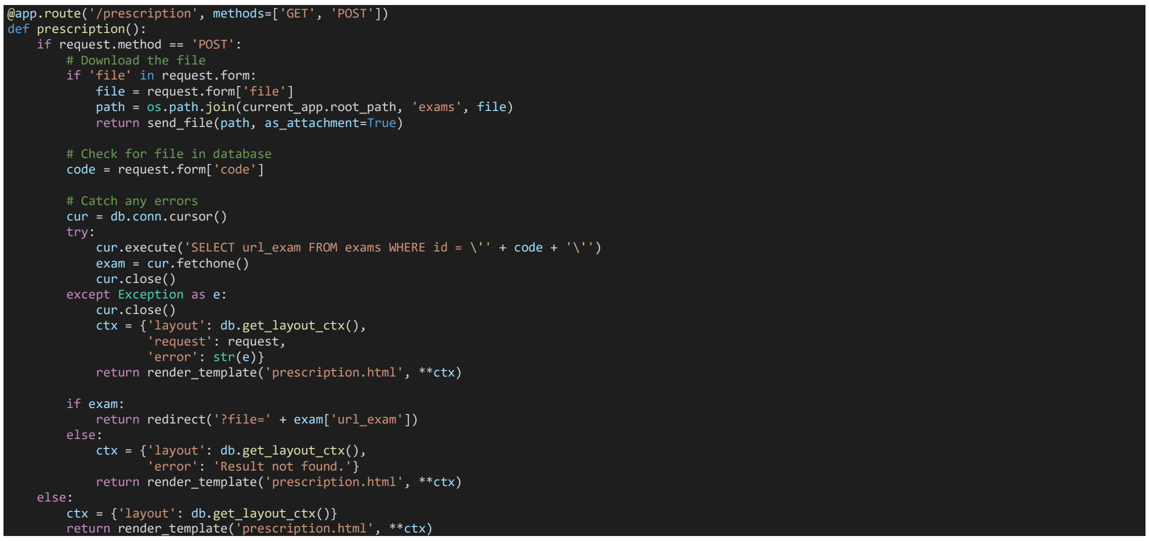
Description

In this vulnerability, the application makes files or directories accessible to unauthorized actors, even though they should not be. Web servers, FTP servers, and similar servers may store a set of files underneath a "root" directory that is accessible to the server's users.

Applications may store sensitive files underneath this root without using access control to limit which users may request those files, if any. Alternately, an application might package multiple files or directories into an archive file, but the application might not exclude sensitive files that are underneath those directories.

Potential Impact

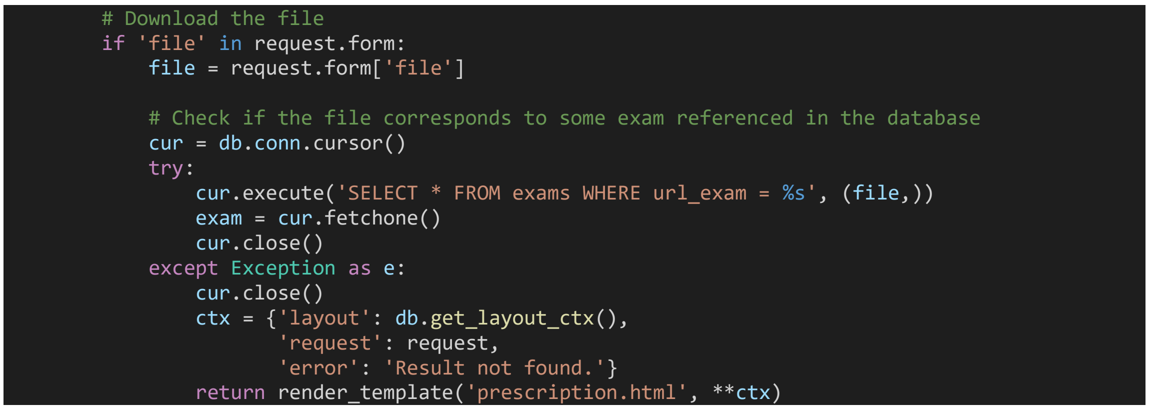
In the insecure application, we can exploit this vulnerability in the exam results page. Changing the argument in the URL (that contains the name of the file of the exam), we can download other people's exams (in the same directory) if we can guess their file names, without knowing the exam code! When accessed, our exam results' page calls the following function:



```python
@app.route('/prescription', methods=['GET', 'POST'])
def prescription():
    if request.method == 'POST':
        # Download the file
        if 'file' in request.form:
            file = request.form['file']
            path = os.path.join(current_app.root_path, 'exams', file)
            return send_file(path, as_attachment=True)

        # Check for file in database
        code = request.form['code']

        # Catch any errors
        cur = db.conn.cursor()
        try:
            cur.execute('SELECT url_exam FROM exams WHERE id = \'' + code + '\'')
            exam = cur.fetchone()
            cur.close()
        except Exception as e:
            cur.close()
            ctx = {'layout': db.get_layout_ctx(),
                   'request': request,
                   'error': str(e)}
            return render_template('prescription.html', **ctx)

        if exam:
            return redirect('?file=' + exam['url_exam'])
        else:
            ctx = {'layout': db.get_layout_ctx(),
                   'error': 'Result not found.'}
            return render_template('prescription.html', **ctx)
    else:
        ctx = {'layout': db.get_layout_ctx()}
        return render_template('prescription.html', **ctx)
```

Correction

In the secure application, when the file name is contained in the URL, we check if that file name corresponds to some exam referenced in the database.



```python
        # Download the file
        if 'file' in request.form:
            file = request.form['file']

            # Check if the file corresponds to some exam referenced in the database
            cur = db.conn.cursor()
            try:
                cur.execute('SELECT * FROM exams WHERE url_exam = %s', (file,))
                exam = cur.fetchone()
                cur.close()
            except Exception as e:
                cur.close()
                ctx = {'layout': db.get_layout_ctx(),
                       'request': request,
                       'error': 'Result not found.'}
                return render_template('prescription.html', **ctx)
```

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

CVSS

- **Score: 7.5**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Description

This is a vulnerability where the application uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory. If the application does not properly neutralize special elements within the pathname, it can cause the pathname to resolve to a location that is outside of the restricted directory.

Potential Impact

Using “..” and “/” separators, the attacker is given the chance to escape outside of the restricted location to access files or directories that are elsewhere on the system.

In the insecure application, we can exploit this vulnerability in the exam results page. The file name argument in the URL lets us insert ‘../’ to go back a page.

Correction

In the secure application, the fixing the previous vulnerability also fixes this one.

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

CVSS

- **Score: 3.7**
- **Vector String:** CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Description

The application performs an operation that triggers an external diagnostic or error message that is not directly generated or controlled by the application, such as an error generated by the programming language interpreter that the software uses. The error can contain sensitive system information.

Potential Impact

In the insecure application, the user is able to see error messages generated by the MySQL database when unsanitized input breaks the syntax of queries. These errors allow the user to see information about the query, tables and data that is stored in the database. An example of this leak of information can be found in the login page. When there is a syntax error in the SQL query, an exception is thrown and the error message gets rendered in the page.

```
        cur = db.conn.cursor()
        try:
            cur.execute(
                'SELECT * FROM users WHERE username = \'' + username + '\' AND password = SHA2(\'' + password + '\', 512)')
            user = cur.fetchone()
            cur.close()
        except Exception as e:
            cur.close()
            ctx = {'layout': db.get_layout_ctx(),
                   'request': request,
                   'error': str(e)}
            return render_template('login.html', **ctx)
```

Correction

In the secure application, we just need to omit this error to the user by showing a vaguer error message.

```
        except Exception as e:
            cur.close()
            ctx = {'layout': db.get_layout_ctx(),
                   'request': request,
                   'error': 'Invalid username or password.'}
            return render_template('login.html', **ctx)
```