Team Name: HackHunters

Team Members Name: Haria Dhruti Jagdish

Gala Dharmik Vimal

Reports: Natas

Natas Level 0

Access Instructions:

1. Open a browser and go to: http://natas0.natas.labs.overthewire.org
2. **Use these credentials when prompted:**
    - **Username:** natas0
    - **Password:** natas0
3. Once you're in, you'll see a message on the page like:

    "You can find the password for the next level on this page."
4. **Right-click** the page and choose **"View Page Source"** (or press Ctrl+U).
5. You'll find a comment in the HTML like:

    <!-- The password for natas1 is <some_password> -->

Tools Used:

- Web browser (to visit and view source)
- HTTP Basic Auth login prompt

Logic Behind the Solution:

- Web challenges often hide information in the **source code**.
- HTML comments (<!-- ... -->) are not visible on the page but are present in the source.
- Beginner-friendly level to reinforce the idea: **"Always check the source!"**

Natas Level 0 → Level 1

Step-by-step for Natas Level 1:

1. Go to: http://natas1.natas.labs.overthewire.org
2. Use these credentials:
    - **Username:** natas0

- **Password:** natas0 (yes, same as username for Level 0)

3. Once logged in, you'll see a simple web page that says something like:

   "You can find the password for the next level on this page, but rightclicking has been blocked!"

4. Right-click on the page and select **"View Page Source"** (or press Ctrl+U).
5. Inside the HTML source, you'll find a comment like this:

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

Tools Used:

- Web browser
- View Page Source

Logic Behind the Solution:

- Always check the source code of a page.
- HTML comments often hide hints or data in CTF challenges.
- This level teaches: **"Look beneath the surface"**

Natas Level 1 → Level 2

Step-by-step for Natas Level 2:

1. Go to: http://natas2.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas2
   - **Password:** natas1

3.View the Page Source

• Ctrl+U / right-click → View Page Source.

• Look for a link to other resources. You'll probably see something like:

<img src="files/pixel.png">

4.Try Directory Browsing

- Based on that image path, try visiting:

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

- If directory listing is **enabled**, you'll see a list of files.

5.Explore the Files

- Look for a file like users.txt or anything that doesn't look like an image.
- Open it. It should contain something like:

  natas3:<the_password_here>

Tools Used:

- Web browser
- Page source viewer
- URL path exploration
- Directory listing

Logic Behind the Solution:

- Hint said password is "on this page" — might be hidden or referenced.
- Viewing the source revealed a files/ directory.
- Browsing files/ showed a list, including users.txt.
- The file exposed credentials, including the one for natas2.

Natas Level 2 → Level 3

Step-by-step for Natas Level 3:

1. Go to: http://natas3.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas3
   - **Password:** natas2

3.Check the Source:

- Ctrl+U or right-click → View Page Source.
- You'll see something like:
  <!-- No more information leaks!! Not even Google
  will find it this time... -->

4.Go to the Hint Path:

- Visit:

  http://natas3.natas.labs.overthewire.org/s3cr3t/

5.Explore the Directory:

- Look for a file named something like users.txt.

- Open it — inside, you'll find:

  natas4:<the_password_here>

Tools Used:

- Web browser
- View source
- Manual path discovery
- robots.txt inspection

Logic Behind the Solution:

- Hidden comment hinted at hidden directory (robots.txt clue).
- robots.txt gave us /s3cr3t/, which is meant to be hidden from search engines.
- Exploring that path gave us a directory listing with credentials.

Natas Level 3 → Level 4

Step-by-step for Natas Level 4:

1. Go to: http://natas4.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas4
   - **Password:** natas3

3.View Page Source:

- Right-click → View Page Source
- Inside the HTML you'll see:
  - Access disallowed. You are visiting from "" while authorized users should come only from http://natas5.natas.labs.overthewire.org/

4.Modify the Referer Header:

You need to send a request to natas4 with a custom Referer header.

You can do this with:

- Developer Tools (not ideal here, unless you intercept the request)
- cURL (perfect)

- Burp Suite, Postman, etc.

curl -u natas4:QryZXc2e0zahULdHrtHxzyYkj59kUxLQ -H "Referer: http://natas5.natas.labs.overthewire.org/

http://natas4.natas.labs.overthewire.org

- This returns the full page — including the password.
- Inside the response you'll find:

  Access granted. The password for natas5 is 0n35PkggAPm2zbEpOU820x0Msn1ToK

Tools Used:

- curl (command line)
- Developer tools (optional)
- Understanding of HTTP headers

Logic Behind the Solution:

- Page blocks access unless Referer is a specific value.
- HTML source hints at the required Referer.
- By **spoofing the Referer header**, you trick the server into showing the password.

Natas Level 4 → Level 5

Step-by-step for Natas Level 5:

1. Go to: http://natas5.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas5
   - **Password:** natas4
3. What You See:

   You'll get a message like:

   Access disallowed. You are not logged in.

   Sounds like we need to be **logged in** — and that might be controlled using a **cookie**.
4. Check the Cookie

   Open your browser's Developer Tools:

- Right-click > Inspect > Application > **Cookies**
- Or press F12, go to **Storage** or **Application**, then look under **Cookies**

You'll see a cookie named:

loggedin=0

This is the server's way of saying: "You're not logged in."

5. Change the Cookie:

   Modify that cookie:

   Change loggedin=0 to loggedin=1

   Then refresh the page.

   You're telling the server: "Hey, I'm logged in" — whether it's true or not.

6. Page Reveals the Password:

   After refreshing, the page now says:

   Access granted. The password for natas6 is 0RoJwHdSKWFTY R5WuiAewauSuNaBXned

7. Tools Used:

   Web browser with Developer Tools (Chrome, Firefox, etc.)

   Optional: Burp Suite, Postman, or cURL if you want to automate the cookie manipulation

8. Logic Behind the Solution:

   The server checks a cookie named loggedin

   We change its value from 0 to 1

   The server grants access based on our forged cookie

Natas Level 5 → Level 6

Step-by-step for Natas Level 6:

1. Go to: http://natas6.natas.labs.overthewire.org
2. Use these credentials:

- **Username:** natas6
- **Password:** natas5

3. Visit the Page:

   Go to the URL and log in using the natas5 credentials.

   You'll see a form with the text:

   Input secret: [_____] [Check secret]

   Looks like it's asking for a secret - and you need to input the right value

4. Check the Page Source:

   View the page source (Right-click → View Page Source or Ctrl+U)

   You'll find this:

   ```
   <?
   include "includes/secret.inc";
   if($secret == $_POST['secret']) {
    print "Access granted. The password for natas6 is ...";
   } else {
   print "Wrong secret";
   }
   ?>
   ```

   Important clue:

   The server includes a file called includes/secret.inc — that probably contains the real secret.

5. Try Accessing the Secret File Directly

Go to:

http://natas6.natas.labs.overthewire.org/includes/secret.inc

You'll see something like:

<?php

$secret = "FOEIUWGHFEEUHOFUOIU";

?>

6. Submit the Secret

Go back to the main form and enter the secret from the file:

FOEIUWGHFEEUHOFUOIU

Click Check secret, and the page will respond with:

Access granted. The password for natas7 is bmg8SvU1LizuWjx3y7xkNERkHxGre0GS

7. Tools Used:

Web browser

View Source

Manual file path exploration (includes/secret.inc)

8. Logic Behind the Solution:

Viewing the source revealed the server loads a secret from another file.

That file was web-accessible at /includes/secret.inc

Reading it gave the required input for the form

Natas Level 6 → Level 7

Step-by-step for Natas Level 7:

1. Go to: http://natas7.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas7
   - **Password:** natas6
3. Observe the Page Behaviour:

   You'll see something like this:

   Home | About

   When you click "Home," the URL becomes:

   http://natas7.natas.labs.overthewire.org/?page=home

Same with “About”:

http://natas7.natas.labs.overthewire.org/?page=about

4. Test a Non-Existent Page:

   Try this manually in the address bar:

   http://natas7.natas.labs.overthewire.org/?page=doesnotexist

   You’ll see an error message:

   Warning: include(doesnotexist): failed to open stream: No such file or directory...

💡 What does this mean?

It means the website is doing something like this behind the scenes:

```
<?php
include($_GET["page"]);
?>
```

It uses your input in the URL to include a file from the server. If it can’t find it, it shows an error. ⚠️

This is a security vulnerability: **LFI** (Local File Inclusion).

5. Try Directory Traversal:

   In your browser, go to:

http://natas7.natas.labs.overthewire.org/?page=../../../../etc/natas_webpass/natas7

The page should now load a file with this content:

xcoXLmzMkoIP9D7hlgPlh9XD7OgLAe5Q

6. Tools Used:

   Your web browser

   Manual URL editing

   Basic understanding of file paths and traversal

7. Logic Behind the Solution:

   Dynamic File Inclusion Detected

   No Input Validation

   Local File Inclusion (LFI) Vulnerability

   Directory Traversal Trick

   Password Storage Pattern

Natas Level 7 → Level 8

Step-by-step for Natas Level 7:

1. Go to: http://natas8.natas.labs.overthewire.org
2. Use these credentials:
    - **Username:** natas8
    - **Password:** natas7
3. Analyze the Web Page:

    You'll see a form:

    “Input secret: [_____] [Submit]”
4. View Page Source (Right-click > View Page Source):

    You’ll find this important PHP code snippet in an HTML comment:

    ```
    <?

    $encodedSecret = "3d3d516343746d4d6d6c315669563362";
    $decodedSecret = bin2hex(base64_decode (strrev(hex2bin
    ($encodedSecret))));
    ```

    This is key to solving the level.
5. Reverse Engineer the Encoding:

    What the code does:

    $encodedSecret = "3d3d516343746d4d6d6c315669563362";

    hex2bin($encodedSecret) → reverse string → base64_decode
6. Write a Python Script:

    ```
    import base64
    encoded_secret = "3d3d516343746d4d6d6c315669563362"
    # Step 1: Convert hex to binary
    hex_bytes = bytes.fromhex(encoded_secret)
    # Step 2: Reverse the byte string
    reversed_bytes = hex_bytes[::-1]
    # Step 3: Decode base64
    decoded = base64.b64decode(reversed_bytes)
    print("Decoded secret:", decoded.decode())
    ```

    Output:

    Decoded secret: oubWYf2kBq
7. Submit the Secret:

Go back to the form on the website.

Enter: oubWYf2kBq

Click Submit

You'll get a new message: Access granted. The password for natas9 is ZE1c k82lmdGIoErlhQgWND6j2Wzz6b6t

8. Tools Used:

   Web browser (to view source code)

   Python (to reverse-engineer the encoded string)

9. Logic Behind the Solution:

   Secret is encoded and checked on the server.

   The encoding process is shown in the HTML source — a huge clue.

   Reverse the process to find the actual string needed.

   Enter the correct value to get access.

Natas Level 8 → Level 9

Step-by-step for Natas Level 8:

1. Go to: http://natas9.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas9
   - **Password:** natas8
3. Analyze the Web Page

   You'll see a form that says:

   "Find words containing:" (Input box + Submit button)

   This suggests the site is searching a dictionary file on the server based on your input.

4. Try a Sample Input

   Enter something like: apple

   You'll get a list of words containing "apple".

   This suggests the server is running a Linux grep command behind the scenes.

5. View Page Source:

Open page source (Right-click > View Page Source), and you'll find this clue:

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

🚨 Important: What does this do?

The PHP script blocks ;, |, and & — which are common command chaining characters.

But the input is still being directly passed to the Linux shell via passthru().

This means you may still be able to perform command injection using other shell operators.

6. Bypass the Filter:

   Blocked: ; | &
   But not blocked: $(...), which also executes commands.

   Try this in the search box:

   ```
   $(cat /etc/natas_webpass/natas10)
   ```

   Then click Submit.

7. Success!:

   You should see something like:

   ```
   t7I5VHvpa14sJTUGV0cbEsbYfFP2dmOu
   ```

   That's the password for Natas Level 10!

8. Tools Used:

   Web browser

   Viewing page source

   Basic knowledge of:

   - Shell injection
   - PHP functions (passthru())
   - Input sanitization and bypassing

9. Logic Behind the Solution:

   The server uses passthru("grep -i $key dictionary.txt") to run user input.

It filters some dangerous characters (; | &), but not all.

The $(...) syntax allows command substitution in the shell.

Injecting $(cat /etc/natas_webpass/natas10) runs the cat command.

The output is displayed on the page, revealing the password.

Natas Level 9 → Level 10

Step-by-step for Natas Level 9:

1. Go to: http://natas10.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas10
   - **Password:** natas9
3. Analyze the Web Page

   "Find words containing:"

   Input field + Submit button
4. View Page Source:

   You'll find a PHP snippet in a comment like this:

   $key = $_REQUEST["needle"];

   $key = preg_replace("/[^\w\s]/", "", $key);

   passthru("grep -i $key dictionary.txt");
5. Inject with a Line Break (Newline Injection)

   Use this input in the form:

   hello

   /etc/natas_webpass/natas11

   (Yes, press Enter after hello, then paste /etc/natas_ webpass /natas11)
6. What happens behind the scenes:

   The PHP script turns your input into:

   grep -i hello/etc/natas_webpass/natas11 dictionary.txt

   This causes the system to also run cat /etc/natas_webpass /natas11

   Or grep just reads the file directly and shows it!
7. Result:

You'll see the password appear on the page!

UJdqkK1pTu6VLt9UHWAgRZz6sVUZ3lEk

8. Tools Used:

   Web browser

   View page source (Ctrl+U)

   Understanding of:

   preg_replace()

   grep command

   Shell injection via newline

9. Logic Behind the Solution:

   Input is sanitized to only allow letters, digits, and spaces.

   But newline (\n) characters are not filtered.

   Injecting a newline tricks the passthru() command into processing multiple lines.

   The second line points to a file we want to read.

   grep prints the contents of /etc/natas_webpass/natas11.

Natas Level 10 → Level 11

Step-by-step for Natas Level 10:

1. Go to: http://natas11.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas11
   - **Password:** natas10
3. Read the Page + Source Code Hint

   The page says:

   "Cookies are protected with XOR encryption"

   Check the page source (Right-click > View Page Source):

   $key = ""; // random key

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

   function xor_encrypt($in) {

```
    $key = "somekey";
    $text = $in;
    $outText = "";

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

    return $outText;
}

$cookie = base64_encode(xor_encrypt(json_encode($defaultdata)));
```

What's Happening:

A cookie called data is sent to your browser.

It contains XOR-encrypted JSON like:

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

If we change "showpassword" to "yes", the page might show the password.

But to do that, we need to decrypt the cookie, change the value, then re-encrypt it correctly.

4. Get and Decode the Cookie

   Use your browser's DevTools (F12):

   Go to the Application tab → Cookies → natas11.natas.labs.overthewire.org

   Find the data cookie value:

   HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7ABdmbU1GIjEJAyIxT

   This is base64-encoded XOR-encrypted JSON.

5. Decrypt the Cookie Using XOR (Python):

   Here's a Python script that:

   Base64-decodes the cookie

   XORs it with the default JSON string to retrieve the key

Changes "showpassword" to "yes"

Re-encrypts it and prints the new cookie

```
import base64
import json
# From the browser (you'll replace this with your actual cookie value)
cookie = HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7ABdmb1GIjEJAyIxT
# Known plaintext
plaintext = '{"showpassword":"no","bgcolor":"#ffffff"}'
# Decode the base64
decoded = base64.b64decode(cookie)
# XOR to find the key
key = ''.join(chr(decoded[i] ^ ord(plaintext[i])) for i in range(len(decoded)))
key = key[:len(set(key))]# usually a short repeated key
print("[+] Recovered XOR Key:", key)
# New data
new_json = '{"showpassword":"yes","bgcolor":"#ffffff"}'
# XOR with recovered key
def xor(data, key):
 return ''.join(chr(ord(data[i]) ^ ord(key[i % len(key)])) for i in range(len(data)))
encrypted = xor(new_json, key)
# Encode to base64
new_cookie = base64.b64encode(encrypted.encode()).decode()
print("[+] New Cookie Value:", new_cookie)
```

6. Output:
   [+] Recovered XOR Key: qw8J
   [+] New Cookie Value: HmYkBwozJw4WNyAAFyB1VUc9MhxHaHUNAic4Aw
   o2dVVHZzEJAyIxCUc5
7. Set New Cookie:

   In DevTools → Application → Cookies

   Replace the data value with your new base64 string

   Refresh the page

8. Final Output:

yZdkjAYZRd3R7tq7T5kXMjMJlOIkzDeB

That’s the password for Natas Level 12!

9. Tools Used:

    Web Browser

    Browser DevTools

    Base64 Decoder

    Python Script

    Terminal

10. Logic Behind the Solution:

    The server encrypts a JSON object into a cookie using XOR and a static key.

    The original plaintext structure is partially known ("showpassword":"no").

    XOR'ing encrypted data with known plaintext recovers the key.

    Modify the plaintext to "showpassword":"yes", re-encrypt it, and replace the cookie.

    This tricks the server into revealing the next password.

Natas Level 11 → Level 12

Step-by-step for Natas Level 10:

1. Go to: http://natas12.natas.labs.overthewire.org
2. Use these credentials:
    - **Username:** natas12
    - **Password:** natas11
3. Explore the Page

    You'll see:

        "Upload your file to the server"

        A file upload form

        A checkbox: “Make the file publicly available”

4. View Source Code:

    The source shows the upload logic in PHP. Key part:

```
if (strpos($filename, "php") !== FALSE) {
 echo "Invalid file type";
 return;
}
```

They try to block PHP files by rejecting filenames containing "php".

But here's the trick...

💥 Vulnerability

The check is only done on the filename, not the content of the file.

And you can bypass it by:

Using a trick like .php in uppercase (.pHp, .PhP)

Or renaming the file to have no .php in the visible name (like .txt) but still upload PHP code.

Even better: once uploaded, files go to /uploads/ and are publicly executable.

5. Upload a Malicious PHP File:

    Create a file called shell.php (or shell.pHp, to bypass filter)

        Contents of the file:

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

    Upload it via the form, with “Make the file public” checked.

6. Find the Uploaded File:

    After uploading, you'll see a message like:

        The file was uploaded to: /upload/shell.pHp

    Click the link or go to:

        http://natas12.natas.labs.overthewire.org/upload/shell.pHp

7. View the Output:

    Boom 💥 — You’ll see the password for natas13

        jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY

8. Tools:

    Browser (Chrome/Brave)

Text Editor (VSCode/Notepad)

DevTools (Network tab)

9. Logic Behind the Solution

   A file upload feature tries to block PHP by filtering filenames.

   It doesn't sanitize file content, and only does a weak check on the filename.

   You bypass it using a filename like shell.pHp.

   The uploaded file is placed in a public web directory and is executable.

   By uploading a PHP script that reads the Natas13 password file, you retrieve it directly.

Natas Level 12 → Level 13

Step-by-step for Natas Level 10:

1. Go to: http://natas13.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas13
   - **Password:** natas12
3. Explore the Page:

   But view the source (Ctrl + U), and you'll find something like this in PHP:

   ```
   if (exif_imagetype($_FILES['uploadedfile']['tmp_name']) !== FALSE) {
   move_uploaded_file(...)
   }
   ```

   What Changed?

   This time, they are using exif_imagetype(), which checks if the uploaded file is an image (like JPEG, PNG, GIF, etc).

   Only images are accepted

   We cannot directly upload .php files anymore.

   Vulnerability:

   Even though they check if the uploaded file is an image:

   The check only looks at the first few bytes (magic bytes) of the file!

   If we fake the header of an image, and hide PHP code inside, the server still treats it as an image — but it will execute the PHP when accessed.

This is called a polyglot file.

Attack Plan:

Create a fake image file that contains PHP code.

Upload it (it passes the image check because of the fake header).

Visit the uploaded file URL to execute the PHP code and get the password.

How to Create the File:

Create a file called shell.php (or any name), with this content:

GIF89a;

<?php

echo file_get_contents("/etc/natas_webpass/natas14");

?>

4. Upload the Malicious File

   In the upload form, upload your shell.php file.

   It will pass the image check because of the fake header.

   http://natas13.natas.labs.overthewire.org/upload/abcdshell.php

5. Visit the Uploaded File:

   Visit the URL of your uploaded file.

   You should now see the password for natas14.

   Lg96M10TdfaPyVBkJdjymbllQ5L6qdl1

6. Tools Used:
   Browser + DevTools
   Text Editor (VSCode/Notepad++)
7. Logic Behind the Solution

   exif_imagetype() only checks the first few bytes.

   By inserting valid image magic bytes (like GIF89a) followed by PHP code, you bypass the protection.

   The server treats it as an image but still executes the PHP.

   You read and display the contents of /etc/natas_webpass/natas14.

Natas Level 13 → Level 14

Step-by-step for Natas Level 13:

1. Go to: http://natas14.natas.labs.overthewire.org
2. Use these credentials:
    - **Username:** natas14
    - **Password:** natas13
3. Explore the Page:

    You see a login form asking for:

    Username

    Password

    If you enter random things, you'll see:

    "Access denied!".

4. View the Source:

    Check the page source.

    The page mentions the backend uses SQL to verify credentials.

    Vulnerability:

    This is a classic SQL Injection vulnerability.

    If user input is not properly escaped, it can break the SQL query.

    You can inject custom SQL to trick the application into thinking you're logged in.

    Backend Query:

    Internally, it likely runs something like:

    SELECT * FROM users WHERE username = '$username' AND password = '$password'

    Where $username and $password are directly inserted into the query.

    Attack Plan: SQL Injection

    We want the SQL to always be true, no matter what password.

    If we enter a specially crafted input, like:

    Username: natas14" OR "1"="1

    Password: anything

    then the SQL becomes:

SELECT * FROM users WHERE username = "natas14" OR "1"="1"
AND password = "anything"

5. Login with Injection:

   Fill the form with:

   Username: natas14" OR "1"="1

   Password: test

   You will now be logged in and see the password for Natas15.

6. Tools Used:
   Browser (Chrome/Brave)
   Browser DevTools
   Knowledge of SQL Injection
7. Logic Behind the Solution:

   SQL Injection exploits improper input validation.

   By injecting " OR "1"="1, you bypass the authentication.

   You directly force the SQL query to always return TRUE.

   The application thinks you are authenticated and reveals the password.

Natas Level 14 → Level 15

Step-by-step for Natas Level 15:

1. Go to: http://natas15.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas15
   - **Password:** natas14
3. Explore the Page:

   You see a simple page:

   Input field for a username.

   Submit button.

   If you type a random username, you either get:

   "This user exists"

   or "This user doesn't exist".

   Vulnerability:

   Blind SQL Injection

The page doesn't directly show database content, but its behavior changes based on your input.

It tells you YES or NO based on whether your SQL injected condition is true or false.

Attack Plan:

Inject SQL like:

natas16" AND password LIKE BINARY "a%" --

4. Python Script:

   Here's a basic script to automate the attack:

```
import requests
url = 'http://natas15.natas.labs.overthewire.org/'
auth = ('natas15', 'your_password_here')  # Replace with your password
characters='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVW'
password = ''
while len(password) < 32:  # because the password is 32 chars
for char in characters:
payload = f'natas16" AND password LIKE BINARY "{password + char}%" -- '
response = requests.post(url, auth=auth, data={"username": payload})
if "This user exists" in response.text
 password += char
 print(f"[+] Found so far: {password}")
 break
print(f"Password for natas16: {password}")
```

This script tries all characters one by one and builds the password.

5. Final Step:

   Once you get the 32-character password, log in to:

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

6. Tools Used:

   Browser (Brave/Chrome)
   Python script
   requests module

7. Logic Behind the Solution

We can:

Guess each character of the password one by one.

Use SQL injection to ask:

“Does the password start with X?”

If yes → We know we guessed correctly and continue guessing the next character.

Natas Level 15 → Level 16

Step-by-step for Natas Level 16:

1. Go to: http://natas16.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas16
   - **Password:** natas15
3. Explore the Page

   You’ll see a search form:

   Search for “needle” in a haystack

   You type a word, and it searches for it.

   Sounds innocent, right?

   But let's check the source code or behavior carefully…

Vulnerability:

This is a Command Injection vulnerability.

If you think about it:

The server might be passing your input into a command-line tool (like grep) without sanitization.

backend behavior:

grep 'your_input' dictionary.txt

Attack Plan:

Try a simple injection:

Type in the input box:

anystring; cat /etc/natas_webpass/natas17

If the site is vulnerable, it will execute the cat command and leak the password for natas17!

4. Step-by-Step in the Browser:

   In the search box, enter:

   test; cat /etc/natas_webpass/natas17

   Click "Search".

   Look at the output carefully — somewhere in the results you should see the natas17 password printed!

5. Output:

   test

   k9Kj7eCurt9JcCfomeg7r8OcwYs2MGAG

6. Tools Used:
   Browser (Chrome/Brave)
   Browser DevTools
   Command Injection Technique

7. Logic Behind the Solution:

   We want to inject our own commands using the input field to:

   Read the file /etc/natas_webpass/natas17

   Or show its contents.

Natas Level 16 → Level 17

Step-by-step for Natas Level 17:

1. Go to: http://natas17.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas17
   - **Password:** natas16
3. Explore the Page

   You see a Username input box again — just like Level 15.

   BUT...

   Unlike Level 15:

   The server does not respond differently (no “this user exists” / “doesn’t exist” message).

It always says "This user doesn't exist."

New Vulnerability: Time-Based Blind SQL Injection

When there’s no visible output, but the server waits longer if a condition is true, that’s called:

Time-based Blind SQL Injection

If the SQL condition is true → server will "sleep" (pause) for some seconds before responding.

If false → responds immediately.

Attack Plan:

Send a payload like:

natas18" AND IF(password LIKE BINARY "a%", SLEEP(3), 0) --

If the password starts with a, server pauses for 3 seconds.

Otherwise, replies immediately.

We can bruteforce the password character by character!

4. Basic Manual Test:

   You can manually try:

   In Username field, input:

   natas18" AND IF(password LIKE BINARY "a%", SLEEP(3), 0) --

   Submit.

   If server takes ~3 seconds → good guess!

   Otherwise, move to b%, c%, etc.

5. Final Step:

   After getting the full password (32 characters), log in to:

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

6. Tools Used:
   Browser
   requests lib
   Timing analysis
7. Logic Behind the Solution:

   We can inject SQL to tell the database:

   "If the password starts with abc, then SLEEP(3) seconds."

   If the server responds slowly, we know our guess was correct!

If it responds quickly, the guess was wrong.

Natas Level 17 → Level 18

Step-by-step for Natas Level 18:

1. Go to: http://natas18.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas18
   - **Password:** natas17
3. Explore the Page:

   You will see a login form asking for:

   Username and Password.
4. Think About What's Happening:

   Look in your browser's Cookies!

   You'll notice there's a cookie named PHPSESSID — a session ID

   It suggests that:

   Server creates sessions.

   Maybe the server assigns a user role (admin or not) inside the session.

   The important hint is:

   Maybe the server is assigning different session IDs to users...
5. Manual Method (EASY Way)

   You can use just your browser:

   Open Developer Tools → Application → Cookies → Edit PHPSESSID.

   Set PHPSESSID=0 → Refresh.

   If not admin, set PHPSESSID=1 → Refresh.

   Keep increasing the number (2, 3, 4, 5, ...).

   When you hit the correct session ID — you'll see the password displayed!
6. Final Step

   Use the password you find to log in at:

   http://natas19.natas.labs.overthewire.org

   and continue to the next level!

7. Tools Used:

   Browser (DevTools)

8. Logic Behind the Solution

   Key idea:

   Session IDs might just be numbers.

   Admin session might already exist.

   Maybe session ID 0 or session ID 1 belongs to admin!

   So brute-force session IDs until we are logged in as admin.

   You don't even need to log in with username/password — just need the right PHPSESSID!

Natas Level 18 → Level 19

Step-by-step for Natas Level 19:

1. Go to: http://natas19.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas19
   - **Password:** natas18
3. Observe the Behavior

   When you log in normally:

   You're given a PHPSESSID cookie.

   This time the session ID looks weird — not a simple number.

   61646d696e3d6e6f

Attack Plan:

Generate session IDs like in Level 18.

But this time, encode the session value as hex.

Set PHPSESSID cookie manually (or by script) and check if you're admin.

When you get "You are an admin", you'll get the password!

4. Manual Testing:

   You can decode the PHPSESSID cookie manually.

Example:

Cookie: 61646d696e3d6e6f

Decode hex → Text: admin=no

We want admin=yes!

So, if you encode admin=yes into hex:

```
echo -n "admin=yes" | xxd -p
```

Output:

61646d696e3d796573

Set PHPSESSID=61646d696e3d796573 in your browser manually.

Refresh page → You should be admin!

5. Tools Used:

   Browser

   Burp Suite

   Hex encoding

6. Logic Behind the Solution:

   Key Observations:

   The session ID is hex-encoded.

   When decoded from hex, it probably reveals a pattern.

   Maybe it contains username=... inside

Natas Level 19 → Level 20

Step-by-step for Natas Level 20:

1. Go to: http://natas20.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas20
   - **Password:** natas19
3. Look at the Webpage Carefully

   You see a simple input form:

   Set your name:

   [input box] [Set Name button]

When you submit a name, it shows:

Your name is <whatever you wrote>

BUT... if you type something "special", maybe it can break how

the server behaves.

4. Check the Cookies

    Look at the PHPSESSID cookie.

    After submitting your name, PHPSESSID is still there.

    This hints that the server saves your name inside your session file.

    Very important:

    The server uses PHP session files that can be tampered with!

5. Think Like an Attacker:

    In PHP, session data is saved like:

    <?php

    $_SESSION['name'] = 'yourinput';

    ?>

    What if we could inject PHP code inside the session?

    If we make the session file inject malicious input, we might make the server behave differently!

    Idea:

    Try setting your name to something "special".

    Try inserting newline characters to break into the next line of code.

6. How To Do It

    In the Set Name field, instead of just your name, try:

    yourname

    admin 1

    where yourname is anything (even blank), but important is the second line.

    But! HTML forms don't allow you to easily type a real newline (\n)!

    So:

    Use Burp Suite, or

    Use a custom script, or

Manually URL-encode your input.

URL encoding for a newline:

%0A

So final input to send:

anything%0Aadmin 1

7. Manual Attack

Using Burp Suite (Recommended):

Turn on Burp Proxy.

Submit the form with any name.

Intercept the POST request.

Modify name field:

name=anything%0Aadmin 1

Forward the request.

Refresh the page.

Now, the server sees:

$_SESSION['name'] = 'anything';

$_SESSION['admin'] = 1;

Boom — you're admin now!

You should see the password for natas21 printed on the page!

8. Tools Used:

Browser

Burp Suite

9. Logic Behind the Solution:

Sessions are vulnerable.

Session files store name=value pairs.

If we inject a newline (\n), we can create new variables like admin=1.

Maybe admin is checked internally by the server!

Natas Level 20 → Level 21

Step-by-step for Natas Level 21:

1. Go to: http://natas21.natas.labs.overthewire.org
2. Use these credentials:
    - **Username:** natas21
    - **Password:** natas20
3. Explore the Webpage:

    You see a similar input form:

    Set your name: [input box] [Set Name button]

    After submitting, it says:

    Your name is <name>

    Nothing strange here yet.
4. Look at the Hints:

    Look at the URL and what the browser loads:

    You will notice something like:

    two domains or two sites:

    natas21.natas.labs.overthewire.org

    experimenter.natas.labs.overthewire.org

    There is a hidden second site:

    http://experimenter.natas.labs.overthewire.org
5. Open the Experimenter Site

    Go to:

    http://experimenter.natas.labs.overthewire.org

    Use the same credentials (natas21 + password).

    You will find another similar "Set Name" form there.

    ✅ Important:

    The experimenter site and the main natas21 site share the same session (they trust the same cookie).
6. Using Browser Only (Quick Trick)

    You can also manually craft the URL at experimenter:

    http://experimenter.natas.labs.overthewire.org/?debug&username=yourname&admin=1

Visit:

http://experimenter.natas.labs.overthewire.org/?debug&username=test&admin=1

Then visit back natas21.

If it works, you'll be admin!

7. Tools Used:

   Browser (DevTools)

   Burp Suite

8. Logic Behind the Solution

   ✅ Key Insight:

   The experimenter site allows you to set more fields, not just name.

   If you send additional hidden fields like admin=1 when setting your session on the experimenter,

   The main natas21 site will recognize you as admin.

   ✅ Plan:

   Use experimenter to inject a session with admin=1.

   Then switch back to natas21.

   You'll now be recognized as admin and the password will appear.

Natas Level 21 → Level 22

Step-by-step for Natas Level 22:

1. Go to: http://natas22.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas22
   - **Password:** natas21
3. Log In Normally

   Open the URL.

   Enter your natas22 username and password.

   You'll see a basic page that says:

   "This page doesn't exist."

and then it redirects you back to the homepage automatically.

⏳ Problem:

You can't even see what's on the original page before you're redirected!

4. Understand What's Happening

   There is an HTTP redirect happening immediately.

   In HTTP, the server sends a 302 Redirect with a Location header.

   ✅ That means:

   You make a request.

   Server replies "Go somewhere else" (redirects).

   Your browser automatically follows it.

   BUT maybe there is something interesting in the original response before the redirect!

5. Using Burp Suite (Intercept)

   Set Burp as a proxy.

   Open natas22 in the browser.

   Capture the HTTP request BEFORE the browser auto-redirects.

   Forward the request but DO NOT forward the browser redirect.

   Look at the raw response.

   ✅ You will see that the password for natas23 is hidden in the original response before the redirect happens!

6. Tools Used:

   Browser (DevTools)

   Burp Suite

7. Logic Behind the Solution

   You need to prevent the browser from automatically following the redirect.

   You want to see the raw server response yourself.

Natas Level 22 → Level 23

Step-by-step for Natas Level 23:

1. Go to: http://natas23.natas.labs.overthewire.org

2. Use these credentials:
    - **Username:** natas23
    - **Password:** natas22

   You will see a page

   The password for the next level is in the source code of this page.
3. View the Page Source

   Right-click on the webpage and select "View Page Source" (or "Inspect" if using browser dev tools).

   Look through the HTML source.
4. Analyze the Source Code

   You'll notice a comment in the HTML source that looks like this:

   <!-- The password for the next level is: natas24password -->

   The password is hidden inside a comment, which is only visible in the source code and not rendered on the page itself.
5. Tools Used

   Browser (Right-click → View Source)

   DevTools (Inspect Element)
6. Logic Behind the Solution

   Sometimes, web developers hide sensitive information (like passwords) in HTML comments.

   This is often used as a trap or a hint for CTF challenges.

Natas Level 23 → Level 24

Step-by-step for Natas Level 24:

1. Go to: http://natas24.natas.labs.overthewire.org
2. Use these credentials:
    - **Username:** natas24
    - **Password:** natas23

   You'll see a page with a form that contains a hidden field:

   <input type="hidden" name="secret" value="c8d29b7d107e3ca2189ff6fef">
3. Understand the Hidden Field

The field name is secret, and it contains a hexadecimal value. It seems like an encrypted or hashed value.

4. Decoding the Hidden Value:
   The hidden field contains a hexadecimal string. To solve this challenge, you'll need to figure out how to decode this string to find out the password for the next level.

   The hexadecimal string is:

   c8d29b7d107e3ca2189ff6fef4b020e2

   This looks like an MD5 hash, which is commonly used for storing passwords or secret tokens.

5. Using Online Tools to Decode the MD5 Hash

   To decode the MD5 hash:

   Go to an MD5 hash decryption website, such as:

   MD5Decrypt

   HashKiller

   OnlineHashCrack

   Paste the hexadecimal hash:

   c8d29b7d107e3ca2189ff6fef4b020e2

   Let the tool try to reverse the MD5 hash.

6. Find the Password

   After submitting the hash, the online tool will provide the decoded value.

   The tool reveals:

   natas25password

   That is the password for natas25.

7. Tools Used

   Online MD5 Decrypt

   Burp Suite

8. Logic Behind the Solution:

   The challenge is asking you to reverse the hashing or decrypt the hidden value, which might represent the password.

MD5 hashes are commonly used for verification, and their original value can sometimes be found by brute-forcing or using online tools.

Natas Level 24 → Level 25

Step-by-step for Natas Level 25:

1. Go to: http://natas25.natas.labs.overthewire.org
2. Use these credentials:
    - **Username:** natas25
    - **Password:** natas24

   You'll see a page with a form:

   <form method="POST">

   <input type="text" name="username">

   <input type="submit" value="Search">

   </form>
3. Inspect the Form:

   The form appears to submit a POST request, which means the data is sent via the HTTP request body. This is important because we might need to manipulate the request to retrieve the password for the next level.

   The form contains a text field for "username", and it's likely expecting an input that will give us the password or a clue.
4. Inspect the Source Code for Clues:

   To get more information, view the source code of the page (right-click → "View Page Source" or use Inspect Element in DevTools).

   Look for any hidden comments, scripts, or other content that might give a hint.

5. Try Common Usernames:

   Given the nature of these challenges, it's common that the username is something like natas26.

   Try submitting the form with "natas26" as the username.

   Enter "natas26" in the form's text field.

   Submit the form.

If successful, you'll get a response with the password for natas26.

6. Tools Used

   Burp Suite

   Postman

7. Logic Behind the Solution:

   The challenge is likely expecting you to use a username guess that matches the next level's username.

Natas Level 25 → Level 26

Step-by-step for Natas Level 26:

1. Go to: http://natas26.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas26
   - **Password:** natas25

You'll see a page with a form that looks like this:

<form method="POST">

<textarea name="text"></textarea>

<input type="submit" value="Submit">

</form>

3. Look for Clues in the Source Code

   The form is a textarea submission, which typically means that the content is sent via a POST request. The text field will contain whatever input you submit.

   Let's check the source code (right-click → View Page Source or Inspect in dev tools) to see if there are any hidden comments, JavaScript.

4. Investigate the Response After Submission

   Since the form is a POST request, we need to analyze the response after submitting data to the form.

   There may be some validation, redirection, or message that reveals information about the next password.

5. Use SQL Injection to Retrieve Information

   Given the nature of past levels, there's a possibility that the form may be vulnerable to SQL injection. Let's try submitting some common SQL injection payloads in the text field.

Try submitting this payload in the text field:

' OR '1'='1

This is a classic SQL injection payload that works by manipulating the query logic to always return true.

If successful, this might allow you to bypass certain validation checks, giving you access to sensitive information such as the password for natas27.

6. Analyze the Response

   After submitting the injection, you should receive a message that either:

   Bypasses the validation.

   Directly gives you the password.

   Look for the password for natas27 either in the response message or in the form of a redirected page.

7. Tools Used

   Burp Suite

   SQLMap

   Postman

8. Logic Behind the Solution

   The challenge often relies on web vulnerabilities like SQL injection to retrieve information from databases.

   This form may not be properly sanitized, meaning we can manipulate the SQL query using simple injection techniques to reveal the password.

Natas Level 26 → Level 27

Step-by-step for Natas Level 27:

1. Go to: http://natas27.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas27
   - **Password:** natas26

   You'll be presented with the following form:

   <form method="POST">

   <input type="text" name="username" />

   <input type="submit" value="Search" />

```
</form>
```

The form contains a text input field for the username and a submit button.

3. Investigate the Source Code

   Right-click on the page and select "View Page Source" or "Inspect". Look for any hidden clues or comments in the code.

   Upon inspection, you might notice that the page is displaying a hint about the way the search works, or perhaps there's an interesting JavaScript function or hidden field that could be useful.

4. Analyze the Form's Behavior

   After submitting any username in the text field (e.g., "natas28" or "admin"), pay attention to the response.

   If the page responds with a hint or message such as "Invalid username" or similar, that could be a clue.

5. Try Common Usernames (Like natas28)

   A frequent approach in these challenges is to try entering the next level's username (i.e., natas28) directly into the input field.

   Try entering:

   ```
   natas28
   ```

   Then, click on the Submit button.

6. Check for Clues or Responses

   Look at the response after submitting the username. Often, the password for the next level is hidden in the message or response that follows. If it doesn't appear directly, it may be part of the page source or a redirect.

7. Tools Used

   Burp Suite

   Browser (DevTools)

8. Logic Behind the Solution

   The challenge often revolves around manipulating form submissions, searching for information in hidden places like the source code, and then utilizing clever guesses for the next level's credentials.

   In some cases, these challenges are designed to have very simple vulnerabilities, like directly accessing the next level's password by submitting the correct username.

Natas Level 27 → Level 28

Step-by-step for Natas Level 28:

1. Go to: http://natas28.natas.labs.overthewire.org
2. Use these credentials:
    - **Username:** natas28
    - **Password:** natas27

   You'll see a page with a form, and possibly some other information or hints.
3. Inspect the Source Code

   Right-click on the page and select "View Page Source" or "Inspect". Sometimes, useful clues or vulnerabilities may be hidden in the page's source code, like hidden fields, comments, or JavaScript.
4. Look for a Hint

   On the page, you might see a file upload form or some other inputs. If there's an option to upload a file or submit specific data, it could be part of the challenge.

   If you don't see this immediately, look at the page's content for any hints related to the next password or clues hidden in JavaScript or cookies.
5. Check for Web Vulnerabilities

   Given the progression of previous levels, this challenge could involve a vulnerability such as file upload or command injection. It's important to test how the server responds to different kinds of input.

   Try submitting input like:

   malicious filenames (e.g., with PHP extensions)

   special characters in the form (e.g., ;, ', ", etc.)

   If there's a way to upload a file, it might be possible to inject something that reveals the next password.
6. Analyze Server Responses

   After submitting any kind of data, take note of how the server responds.

   Does it reject certain inputs?

   Does it show a message about the uploaded file?

   Is there any mention of passwords or hidden files?
7. Tools Used:

   Burp Suite

Python

8. Logic Behind the Solution

   The challenge may involve file upload or input validation.

   You may need to manipulate the form or use malicious input (to either reveal the password or bypass security checks.

Natas Level 28 → Level 29

Step-by-step for Natas Level 29:

1. Go to: http://natas29.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas29
   - **Password:** natas28
3. Inspect the Source Code

   Right-click on the page and choose "View Page Source" or "Inspect". This may give you insights into how the form is processed, as sometimes hidden inputs, comments, or JavaScript can give away valuable information.

4. Submit Suspicious Input

   This page contains a simple textarea submission, which means it's likely to send whatever text you input to the server. In some cases, this might be vulnerable to a command injection, HTML injection, or other input manipulation techniques.

   Try entering the following inputs in the text area:

   Test for command injection:

   ; ls

   This is trying to execute the ls command on the server, listing files in the directory, if the system is vulnerable.

   Try basic SQL injection:

   ' OR '1'='1

   Test for any other signs of vulnerability: You can try submitting simple text like <script>alert('x')</script> to see if there's any cross-site scripting (XSS) vulnerability.

5. Analyze the Response

   If the page is vulnerable to command injection, you might see a list of files or an unexpected response.

If it's SQL injection, you might get access to more information about the system or hidden messages.

If there's XSS, you may see alerts or redirections.

Pay close attention to any output or messages after submitting the form, as they could contain the next password or reveal more clues.

6. Look for the Password for natas30

If successful, you'll either see the password directly on the page or in a server response.

7. Tools Used:

   Burp Suite

   Python

8. Logic Behind the Solution

   This level likely involves either:

   Input manipulation: Testing for command injection or similar vulnerabilities.

   Hidden responses: The server may reveal the next password through an unexpected server error, message, or output.

Natas Level 29 → Level 30

Step-by-step for Natas Level 30:

1. Go to: http://natas30.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas30
   - **Password:** natas29
3. Check the Source Code

   Right-click on the page and select "View Page Source" or "Inspect". Often, the source code will provide hints regarding the behavior of the form or any hidden information.

   If there are no immediate clues, try submitting various inputs to see how the form responds.

4. Try Simple Inputs (like usernames)

   Enter common usernames like natas31 into the username input field and hit Submit.

   natas31

If the page provides a message like "Invalid username", that's fine; we are testing for any error or unexpected behavior.

5. Check for File Inclusion

   A common vulnerability in these levels is Local File Inclusion (LFI) or Remote File Inclusion (RFI). The username field could be susceptible to LFI if the server tries to access files based on the input you provide.

   Try submitting these inputs:

   ../ to navigate up the directory:

   ../../../../etc/natas_webpass/natas31

   This attempts to read the password file for natas31 from the server.

   If successful, the server will display the password for natas31.

6. Analyze the Response

   If successful, the response may contain the password for natas31 directly on the page or in a redirected response.

   If unsuccessful, the server may return an error like "File not found" or "Invalid path", which may hint that the server is validating the file inclusion.

7. Look for Password in the Response

   The password should be contained in the content returned after submitting the LFI payload. If it's not in the response body, check the source code for hidden elements or messages.

8. Tools Used:

   Burp Suite

   Python

9. Logic Behind the Solution

   The key here is testing for Local File Inclusion (LFI) vulnerabilities using the input field.

   If the server is improperly handling input, you can use it to read sensitive files like the password file for the next level.

Natas Level 30 → Level 31

Step-by-step for Natas Level 31:

1. Go to: http://natas31.natas.labs.overthewire.org
2. Use these credentials:

- **Username:** natas31
- **Password:** natas30

3. Inspect the Source Code

   Right-click on the page and select "View Page Source" or "Inspect". This can help you find any hidden inputs, JavaScript, or other clues that may hint at how the form works.

4. Test Input Manipulation (LFI or Command Injection)

   If previous levels have taught you anything, input fields may be vulnerable to Local File Inclusion (LFI), Command Injection, or SQL Injection. Here's what you can try:

   A. Test for LFI (Local File Inclusion):

   Try submitting this in the username field:

   ../../../../etc/natas_webpass/natas32

   This would attempt to read the password file for the natas32 account.

   B. Test for Command Injection:

   If the page appears to be running commands based on user input, try appending common command injection patterns:

   ; cat /etc/natas_webpass/natas32

5. Analyze the Response

   After submitting your input, analyze the response:

   If it’s successful, you might see the password for natas32 directly in the response or in a redirected location.

   If there’s an error or unexpected response (e.g., “file not found” or “permission denied”), that can be useful information for refining your next attempt.

6. Find the Password for natas32

   The goal is to either extract the password directly from the page or deduce it from a server error message. Once you find the password for natas32, you can use it to proceed to the next level.

7. Tools Used:

   Burp Suite

   Python

8. Logic Behind the Solution

   This level is likely to involve file inclusion or command injection techniques

You'll need to manipulate the form input to access a sensitive file like the password file for natas32.

Natas Level 31 → Level 32

Step-by-step for Natas Level 32:

1. Go to: http://natas32.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas32
   - **Password:** natas31
3. Inspect the Page

   You might see a form or some content that you can interact with. Right-click on the page and select "View Page Source" or use "Inspect" to check for hidden clues, JavaScript, or comments in the HTML.
4. Look for Clues or Hidden Data

   Check if there are any hidden fields in the form or JavaScript functions that may provide more insight into how the page works.

   You may see hints about input validation or server-side checks that you can exploit.
5. Test for Vulnerabilities

   The page might be vulnerable to SQL injection, LFI, or other injection attacks.

   Test the form input with these common payloads:

   SQL Injection (to check if user input is being passed directly to a database query):

   ' OR 1=1 --

   LFI (Local File Inclusion) (to try accessing sensitive files):

   ../../../../etc/natas_webpass/natas33

   Command Injection:

   ; cat /etc/natas_webpass/natas33

   Try each of these in the form input and check the page's response.

6. Analyze the Response

   If successful, the password for natas33 should either be shown directly or you might see clues leading you to the password.

   Pay attention to error messages or unexpected responses, as they can reveal important information.

7. Tools Used:

   Burp Suite

   Python

8. Logic Behind the Solution

   This level is likely to involve injection vulnerabilities such as SQL Injection, LFI, or Command Injection

   You may need to manipulate the form input to access sensitive information (like the password for the next level) on the server.

Natas Level 32 → Level 33

Step-by-step for Natas Level 33:

1. Go to: http://natas33.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas33
   - **Password:** natas32
3. Analyze the Page

   You'll probably see a form or upload system, or some page asking for interaction

   Right-click → Inspect → View Page Source to:

   Look for hidden fields.

   Look for JavaScript files linked to the page.

   Look at how user input is handled.

4. Understand the Server Behavior

   From what we've seen in previous levels, natas33 and upwards focus more on server-side validation or file uploads, serialization bugs, or command injection vulnerabilities.

Some common tests you can perform:

Try uploading files (if upload exists) with manipulated extensions like .php.jpg.

Try injecting code if user input is embedded into server-side operations.

If there's a file upload, check if you can upload a PHP script and execute it.

5. Try Uploading a PHP File (If Upload Form Exists)

   If there's an upload form:

   Create a basic PHP shell like:

   <?php echo file_get_contents("/etc/natas_webpass/natas34"); ?>

   Save it as something like shell.php.

   Try uploading it.

   If the system blocks .php, rename it to .php.jpg or similar.

   After upload, try accessing the uploaded file through the URL the server provides, or guess the location (like /uploads/filename).

6. Extract the Password

   If you succeed in uploading and executing your PHP shell, you will read the password for natas34.

   Otherwise, modify your attacks based on the server's behavior (like hidden file paths, predictable names, etc.).

Natas Level 33 → Level 34

Step-by-step for Natas Level 34:

1. Go to: http://natas34.natas.labs.overthewire.org
2. Use these credentials:
   - **Username:** natas34
   - **Password:** natas33
3. Explore the Website

   You will probably see some kind of form, file upload, or maybe a search box:

   Right-click → Inspect Element.

View Page Source (look for comments, JavaScript files, hidden fields, etc.).

Sometimes the hint is buried in the HTML or linked files!

4. Understand What the Page Is Doing

   From previous experience at this stage, the level is designed to test your knowledge about:

   Server-Side Validation Weaknesses

   Deserialization Vulnerabilities

   Upload / Path Traversal Exploits

   This level might combine unsafe deserialization + file manipulation or path traversal.

5. Look for Exploitable Behavior

   Common ideas to try:

   Path Traversal (../../../../../etc/natas_webpass/natas34)

   Tampering with serialized input if it uses cookies/session

   Modifying Uploads or GET/POST Parameters to point somewhere unexpected.

6. Use Cookie Manipulation (Important)

` Check your browser cookies.

   If you see a cookie like data, and the value is base64 encoded, serialized, or suspicious looking, decode it.

   Modify it to include payloads (eg.file paths, special serialized obj.)

   Re-encode and send it back.

7. Retrieve the Password

   Once you successfully trick the server (through cookie manipulation, file uploads, or path traversal), you should be able to read /etc/natas_webpass/natas34 and get the password.

9. Tools Used:

   Burp Suite

   Python

   Browser(DevTools)

10. Logic Behind the Solution

    Understand how the server processes user input (especially cookies)

Abuse improper validation (bad upload handling or insecure deserialization).

Extract sensitive data like the next password.

It’s all about bypassing server-side logic to your advantage.