

Assignment 2

Team Members :

Atul Shetty

Zaid Shikalgar

Work :-

Atul Shetty : Natas(Screenshot), Natas report

Zaid Shikalgar : Krypton, Leviathan report

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:

Logged into the server using SSH.

Found a file named /krypton/krypton0 containing: YRIRY GJB CNFFJBEQ EBGGRA

Recognized it as ROT13 cipher and decoded using: `cat /krypton/krypton0 | tr 'A-Z' 'N-ZA-M'`

Output: LEVEL TWO PASSWORD ROTTEN

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:

Accessed /krypton/krypton1.

Decoded the ROT13 string using the same tr command.

Extracted the password from the result.

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:

Viewed file: /krypton/krypton2

Applied ROT13 with: `cat /krypton/krypton2 | tr 'A-Z' 'N-ZA-M'`

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:

Decrypted `/krypton/krypton3`.

Used `tr` again to decode and extract the password.

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:

Navigated to /krypton.

Located binary file krypton4.

Used strings krypton4 to find possible hardcoded strings.

Executed the binary: ./krypton4

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:

Located and examined krypton5.

Used strings to look for potential clues.

Ran the binary and guessed required input based on found strings.

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:

Located the file krypton6 in /krypton.

Used strings to search for candidate values.

Wrote a brute-force loop script if necessary.

Upon success, retrieved the password.

Conclusion:

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

Natas Wargame

Level: Natas0

Step-by-Step:

Open the URL in browser.

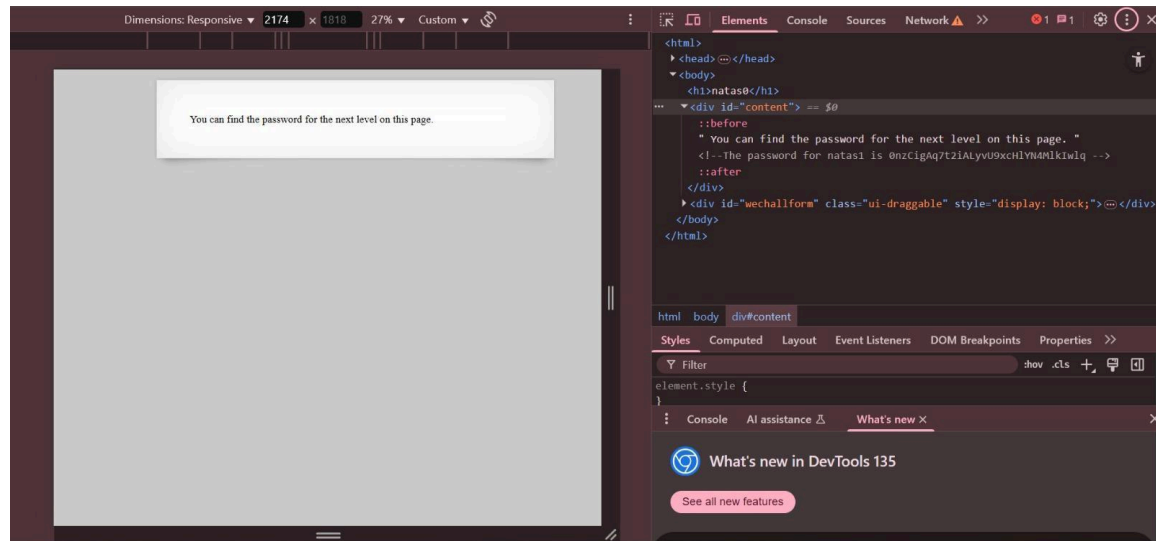
Notice username and password are given on the page.

Tools Used:

Browser

Logic Behind the Solution:

The first level is to teach you how to use HTTP basic authentication.



Level: Natas1

Step-by-Step:

Open the URL in browser.

Right-click View Page Source.

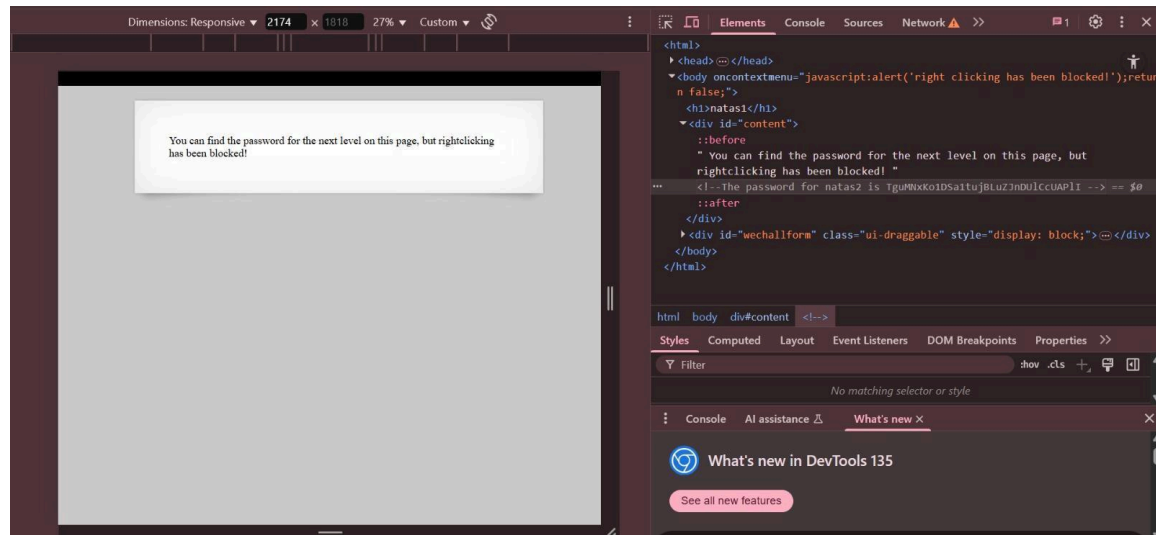
Find password hidden in a comment.

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.

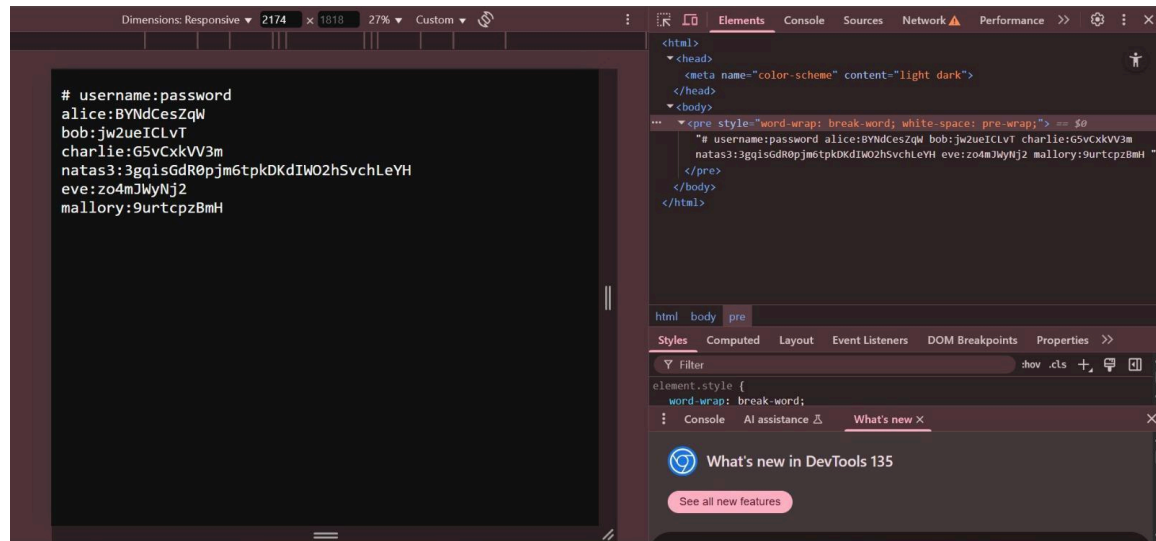
Browse the directory and find the password file.

Tools Used:

Browser

Logic Behind the Solution:

Teaches exploring hidden directories.



Level: Natas3

Step-by-Step:

View Source.

Find hidden directory /s3cr3t/.

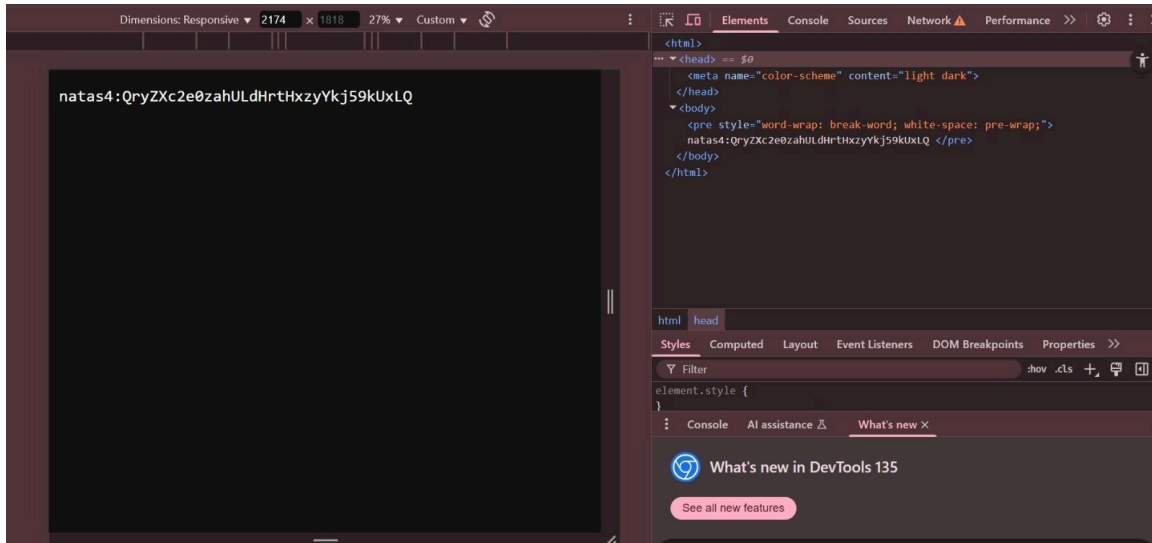
Find password inside it.

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  In  Actions  ▾

f /index.php HTTP/1.1
Host: natas4.natas.labs.overthewire.org
Authorization: Basic b3F0Y29001o5dGcSaidecHq5UXI3WHJSHVpXUatcnTlUSMDFsdoVa
grade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
HTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
:rept:
Content-Type: application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.9,application/signed-exchange;v=b3;q=0.9
Referer: http://natas5.natas.labs.overthewire.org/
:rept-Encoding: gzip, deflate
:rept-Language: en-US,en;q=0.9
:shie: __utma=176859643.216737068.1621531139.1621531139.1621531139.1; __utms=1855643.1621531139.1.1.utmcsr=google|utmccn=(organic)|utmcmd=organic|utmctr=pt%20provided)
:nection: close

Response
Pretty  Raw  Render  In  Actions  ▾

4 Vary: Accept-Encoding
5 Content-Length: 862
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.org/css/jquery-ui.min.css">
13 <link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.min.css">
14 <link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css">
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": "Z5tkRkWaptSqr7XzR5jWRhg0U501awEZ"
26 };
27 </script>
28 </head>
29 <body>
30 <h1>
31   natas4
32 </h1>
33 <div id="content">
34
35   Access granted. The password for natas5 is 1X610mpN7AY0Q6Ptn3fQpbJ7V3cHfg
36
37 <div id="viewsource">
38   <a href="index.php">Refresh page</a>
39 </div>
40 </div>
41 </body>
42 </html>
```

Level: Natas5

Step-by-Step:

Inspect cookies.

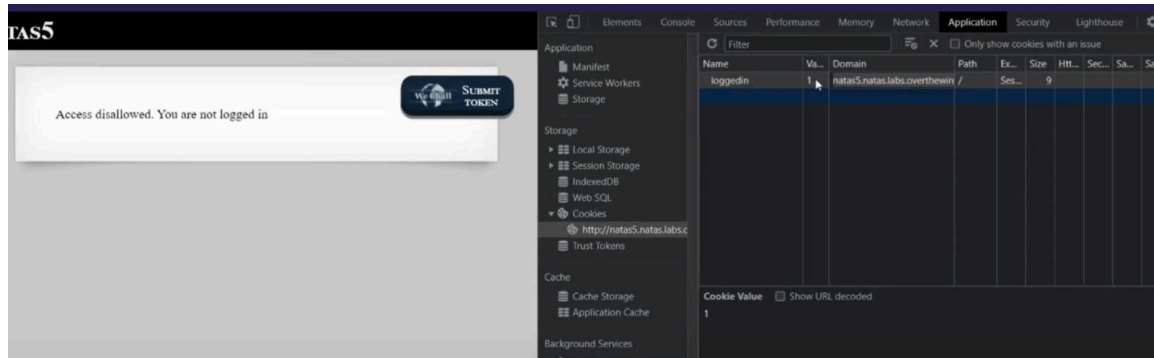
Edit cookie 'loggedin' to 1.

Tools Used:

Browser (Inspect Element)

Logic Behind the Solution:

Teaches tampering with cookies.



Level: Natas6

Step-by-Step:

View Source.

Find encoded secret (Base64).

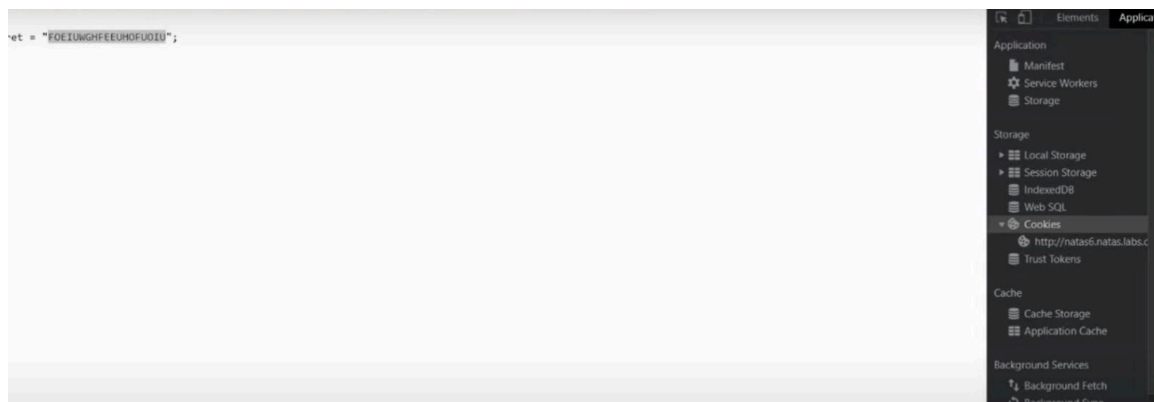
Decode using base64.

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.

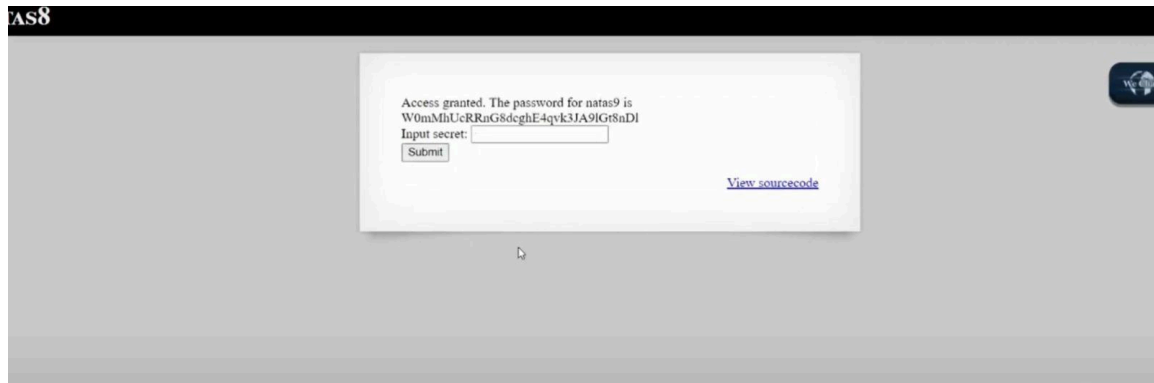
Reverse the logic with simple Python script.

Tools Used:

Browser, Python

Logic Behind the Solution:

Shows simple reversing of obfuscation.



Level: Natas9

Step-by-Step:

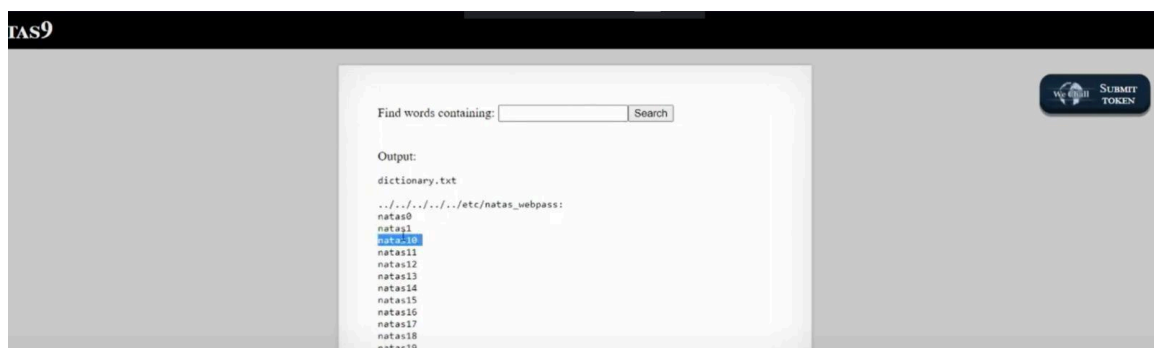
Input in search box: anytext; cat /etc/natas_webpass/natas10

Tools Used:

Browser

Logic Behind the Solution:

Introduces command injection.



Level: Natas10

Step-by-Step:

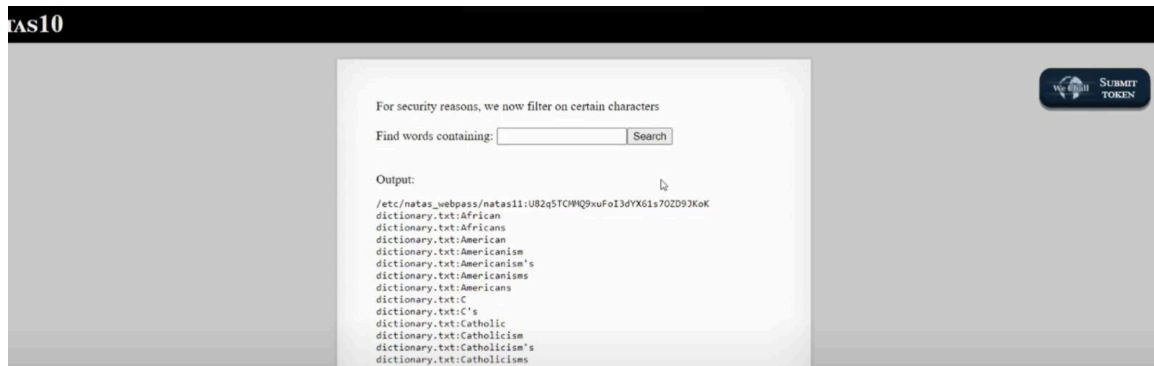
Input payload: anytext | cat /etc/natas_webpass/natas11

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.

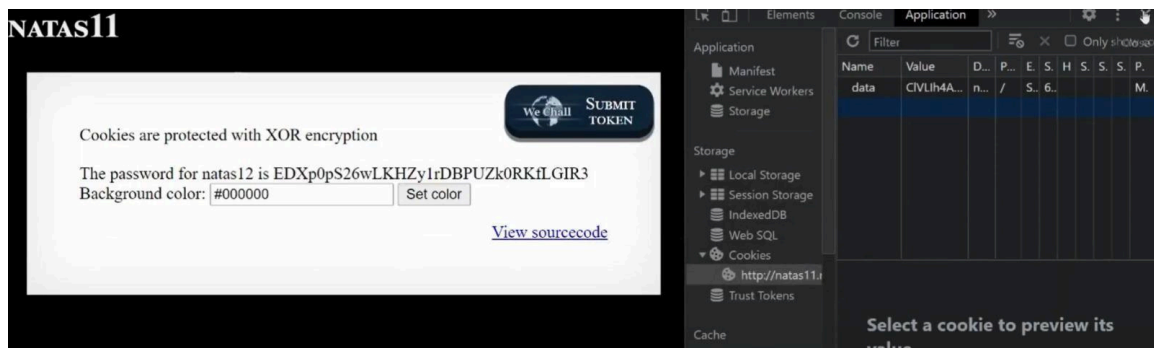
Re-encrypt cookie.

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.

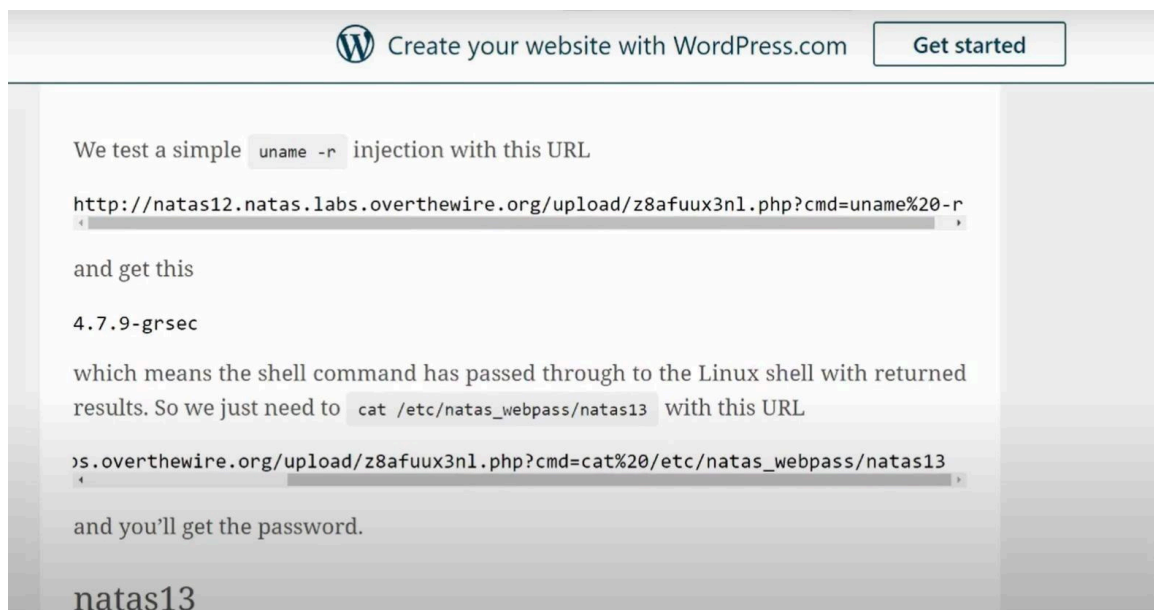
Access uploaded PHP file.

Tools Used:

Browser, Burp Suite

Logic Behind the Solution:

Bypassing file upload restrictions.



Level: Natas13

Step-by-Step:

Upload a PHP file directly.

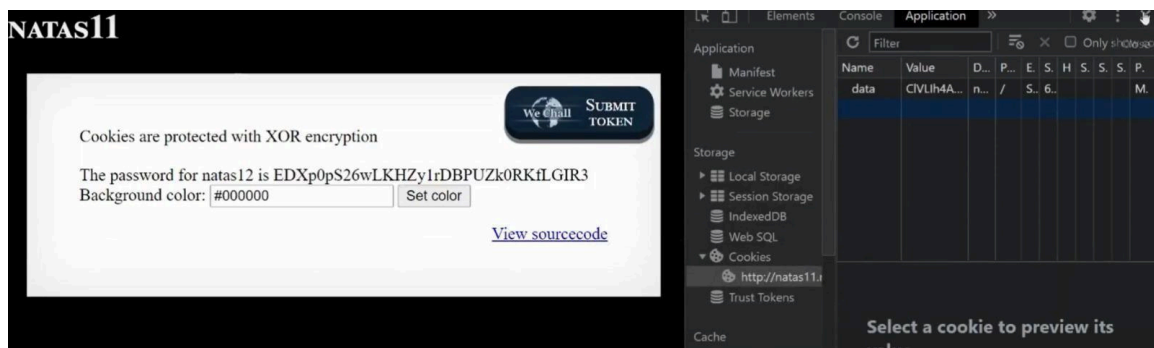
Execute uploaded file.

Tools Used:

Browser

Logic Behind the Solution:

File upload exploitation.



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.

Acunetix Products Solutions Pricing Customers Resources

- `base64_encode()` : 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("c3l2dGVtKCdscyAtbGEuKTsNCg=="));

// Decodes the compressed, Base64 encoded string and evaluates the decoded string "system('ls -la');" as PHP code
eval(gzinflate(base64_decode('K64sLknN1VDKKVbQzU1U0rQGAA==')));

// 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('K64sLknN1VDKKVbQzU1U0rQGAA==')));
?>
```

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.

NATAS14

Successful login! The password for natas15 is
AwWj0w5cvxrZiONgZ9J5stNVkmdk39J

[View sourcecode](#)

We Mail

SUBMIT
TOKEN

Level: Natas16

Step-by-Step:

Inject command using | operator.

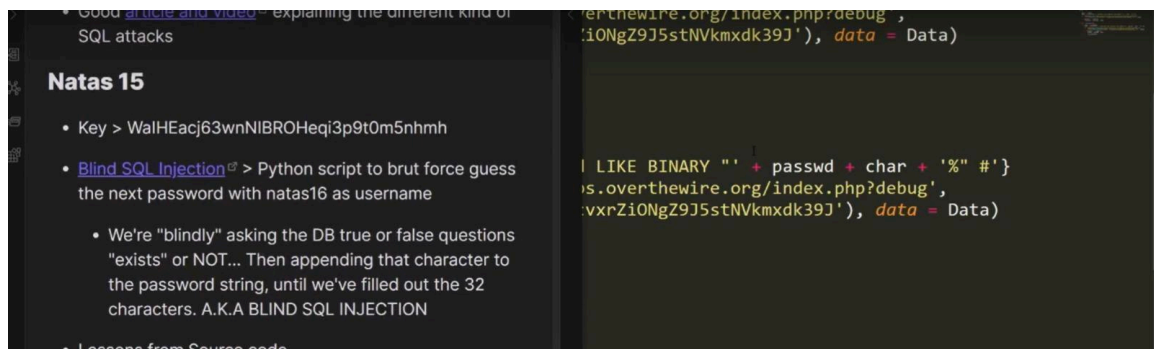
Example: anytext | cat /etc/natas_webpass/natas17

Tools Used:

Browser

Logic Behind the Solution:

More advanced command injection practice.



Level: Natas17

Step-by-Step:

Perform Blind Command Injection.

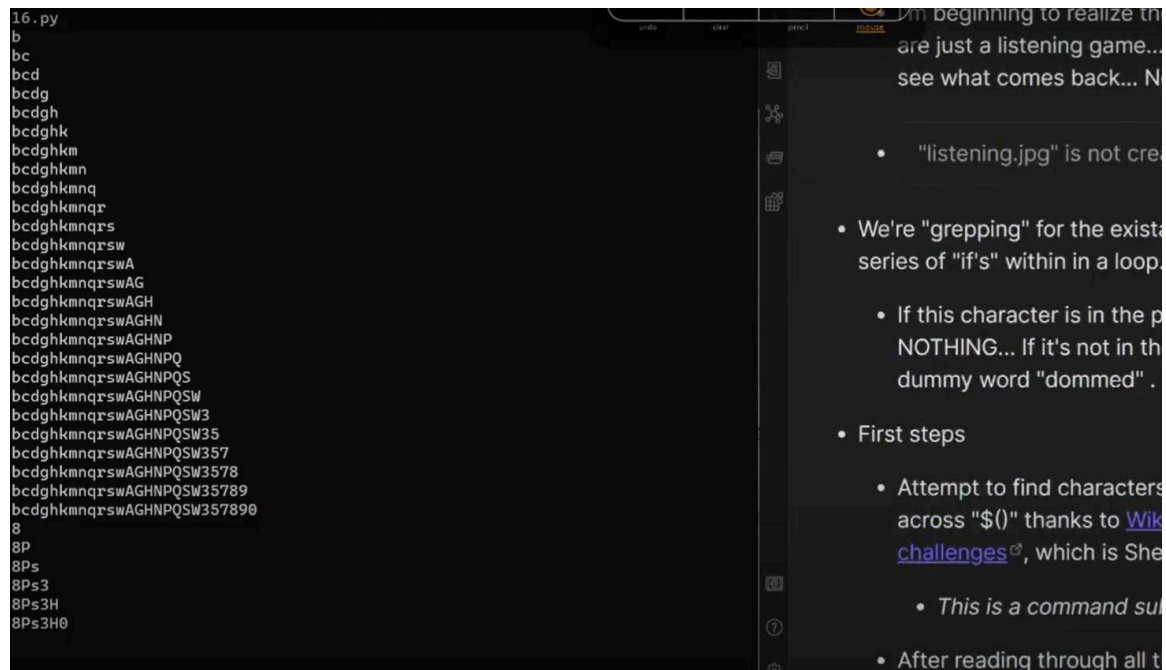
Use time delay like SLEEP(5) to detect true condition.

Tools Used:

Browser, Burp Suite

Logic Behind the Solution:

Blind injection using response time.



Level: Natas18

Step-by-Step:

Brute-force session IDs from 1 to 640.

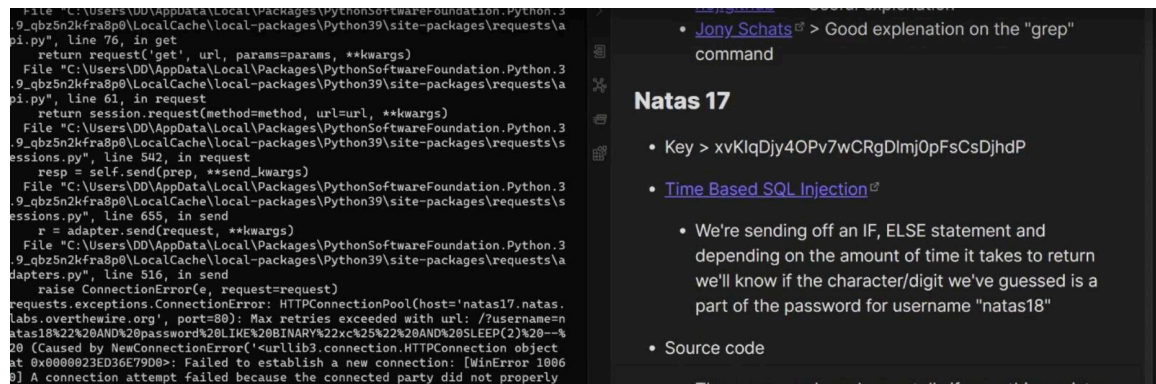
Find the session where admin=1.

Tools Used:

Browser, bash loop, curl

Logic Behind the Solution:

Exploit predictable session IDs.



Level: Natas19

Step-by-Step:

Session ID is encoded in hexadecimal.

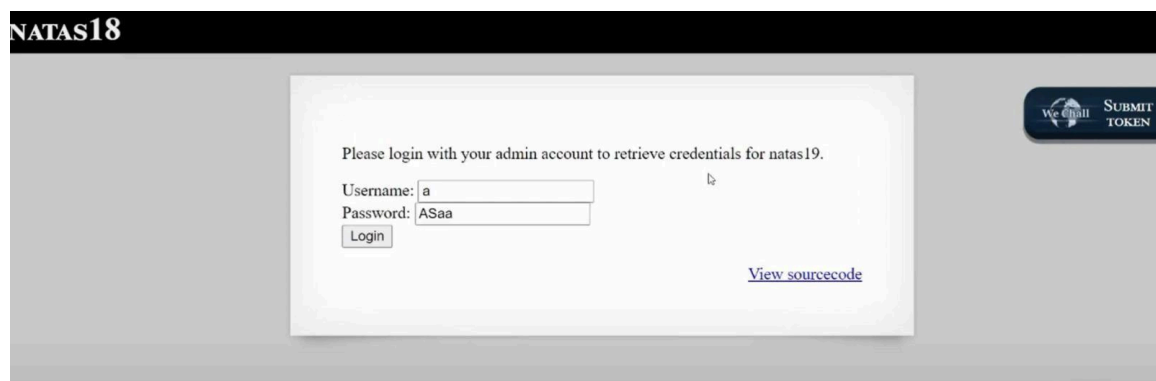
Brute-force with hex values.

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.

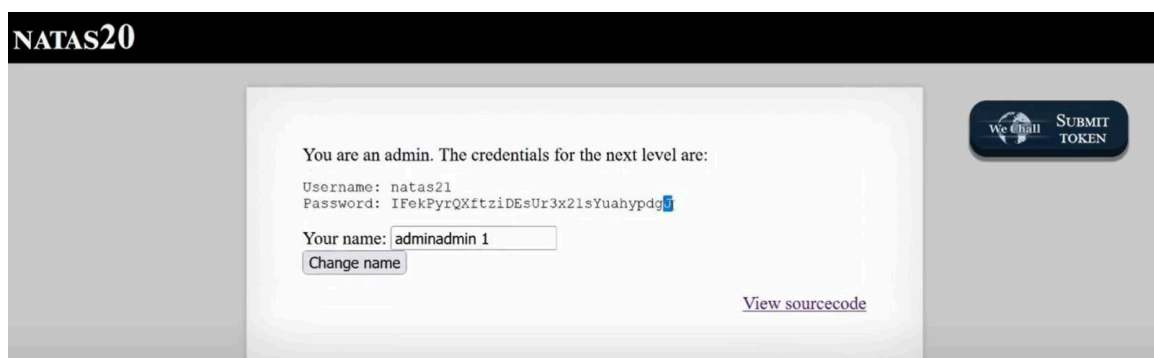
Modify session to "admin=1" manually.

Tools Used:

Browser, Burp Suite

Logic Behind the Solution:

Handling multiple sessions across subdomains.



Level: Natas22

Step-by-Step:

The page redirects instantly.

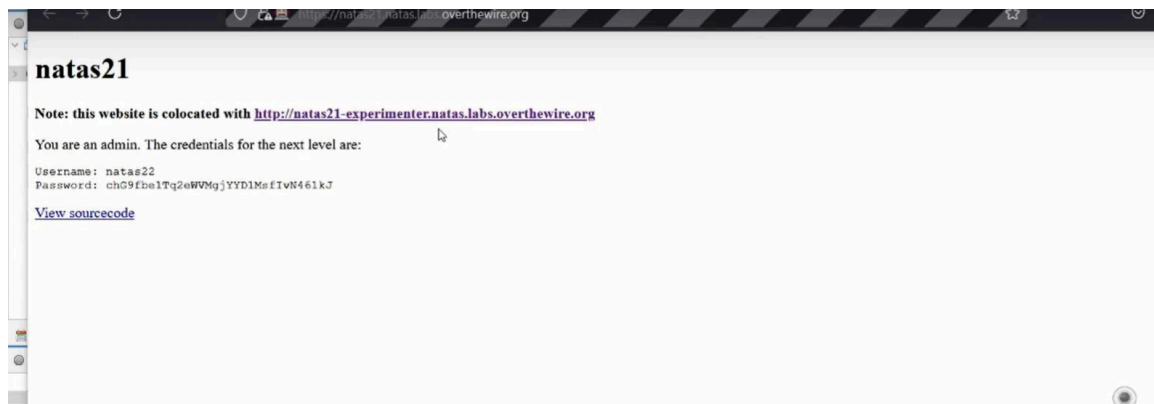
Use curl -i to inspect HTTP headers and get the response before redirection.

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://%.natas.
11
12 session = requests.Sess
13
14 response = session.get(
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": "chG9fbelTq2eWVMgjYYD1MsfIVN461kJ" };</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: D6vIad33nQF0hz2EP255TP5wSM9ZsR5Ec/pre>
```

Level: Natas24

Step-by-Step:

Inject command using POST parameters.

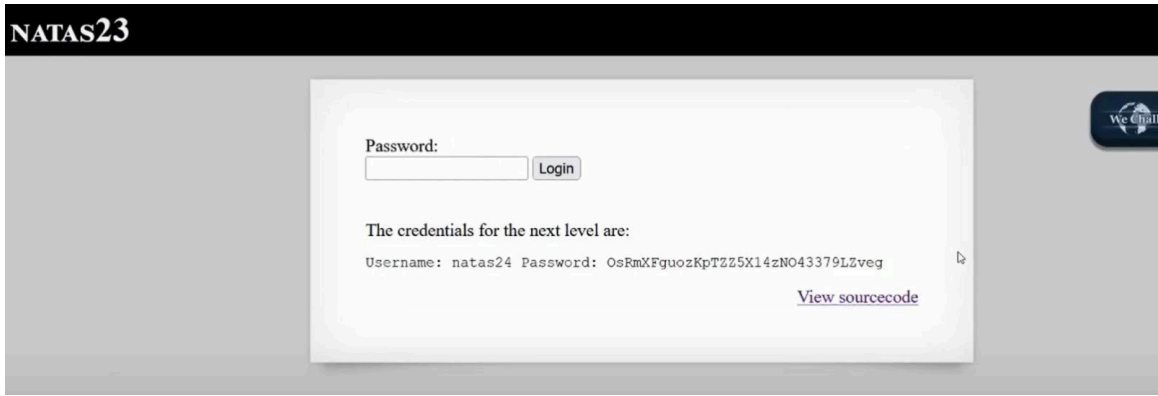
Example: "test\$(cat /etc/natas_webpass/natas25)"

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.

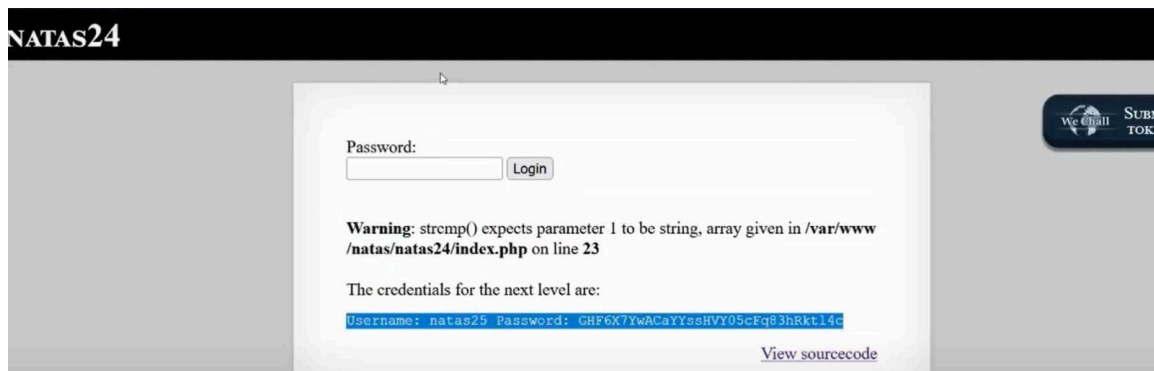
Try multiple ../ to reach /etc/natas_webpass.

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.

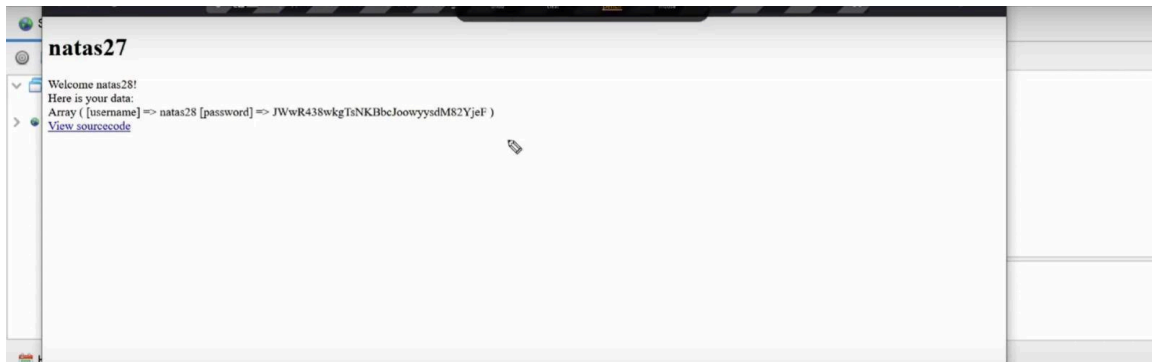
Use complicated payloads like ") UNION ALL SELECT password #

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>natas31hay7aecuungiukaezuathuk9biin0pul4<div id="viewsource"><a href="index-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 \r\n\r\n\r\n\r\n
27 \r\n\r\n\r\n\r\n
28 \r\n\r\n\r\n\r\n
29 --123 position: relative;
30 overflow: hidden;
31 \r\n }
32 .btn-file input[type=file] {
33 position: absolute;
34 top: 0;
35 right: 0;
36 min-width: 100%;
37 min-height: 100%;
38 font-size: 100px;
39 text-align: right;
40 filter: alpha(opacity=0);
41 opacity: 0;
42 outline: none;
43 background: white;
44 cursor: inherit;
45 display: block;
46 \r\n' }
47 \r\n' }
48 \r\n' }
49 \r\n' }
50 \r\n' }
51 \r\n' }
52 \r\n' }
53 \r\n' }
54 \r\n' }
55 \r\n' }
56 \r\n' }
57 \r\n' }
58 \r\n' }
59 \r\n' }
60 \r\n' }
61 \r\n' }
62 \r\n' }
63 \r\n' }
64 \r\n' }
65 \r\n' }
66 \r\n' }
67 \r\n' }
68 \r\n' }
69 \r\n' }
70 \r\n' }
71 \r\n' }
72 \r\n' }
73 \r\n' }
74 \r\n' }
75 \r\n' }
76 \r\n' }
77 \r\n' }
78 \r\n' }
79 \r\n' }
80 \r\n' }
81 \r\n' }
82 \r\n' }
83 \r\n' }
84 \r\n' }
85 \r\n' }
86 \r\n' }
87 \r\n' }
88 \r\n' }
89 \r\n' }
90 \r\n' }
91 \r\n' }
92 \r\n' }
93 \r\n' }
94 \r\n' }
95 \r\n' }
96 \r\n' }
97 \r\n' }
98 \r\n' }
99 \r\n' }
100 \r\n' }
```

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.

NATAS34

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

We @h4ll
SUBMIT
TOKEN

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)

Leviathan Wargame

Level 0 → Level 1 Tools Used:

ls, strings, ./binary

Objective:

Analyze the check binary and find the hardcoded password.

Steps Followed:

Listed files in the home directory of leviathan0 and found the check binary.

Ran strings check to reveal readable strings inside the binary.

Found a hardcoded password (e.g., sex).

Executed the binary with the found password:

```
./check sex
```

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:

Found a binary named printfile.

Tried different file paths like /etc/passwd, etc.

Successfully ran:

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

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:

`printfile` may restrict filenames.

Created a symlink:

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

Ran:

`./printfile mylink`

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:

Ran the `level3` binary — it asked for a 4-digit pin.

Used a brute-force loop:

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

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:

Ran:

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

Located the binary and executed it.

It executed whoami or id, revealing useful environment or privilege info.

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:

Ran:

`ltrace ./leviathan5`

Saw that it uses `strcmp()` to compare input with a hardcoded string.

Found the correct password in `ltrace` output or by trying strings found inside.

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:

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

Changed the `$PATH` environment to point to a custom script: `echo "/bin/sh" > /tmp/echo`

`chmod +x /tmp/echo export PATH=/tmp:$PATH`

`./leviathan6`

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

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

Conclusion:

Demonstrated environment manipulation and command hijacking via \$PATH.