

**PENUGASAN CYBERSECURITY  
OEMOEM 2024**



**Farhan Adiwidya Pradana**  
**Gmail:** [farhanadiwidya@gmail.com](mailto:farhanadiwidya@gmail.com)/  
farhanadiwidyapradana@mail.ugm.ac.id  
**Github:** Excovatedmind (current)

## Level 0-5 OverTheWire Bandit

### 1. Level 0

Goal: Connect to the Bandit server using SSH

Details:

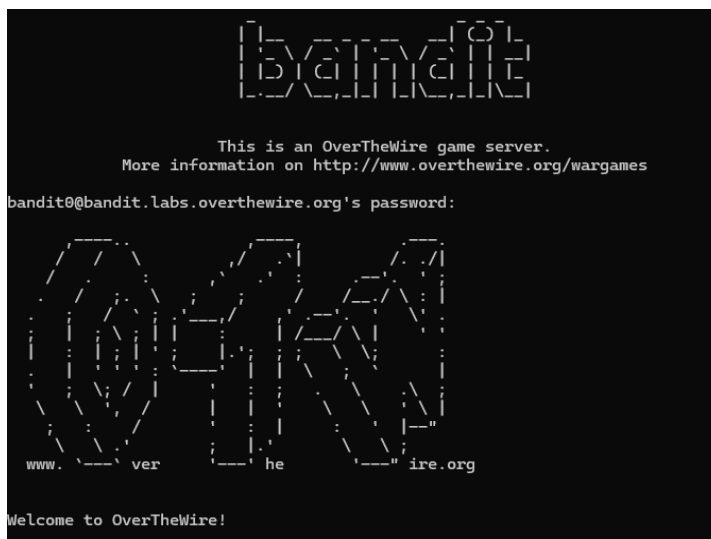
- Server: bandit.labs.overthewire.org
- Port: 2220
- Username: bandit0
- Password: bandit0

Commands used:

#### Connecting to the server

```
ssh bandit0@bandit.labs.overthewire.org -p 2220
```

Screenshot:



Brief Explanation:

- The ssh command is used to establish a secure connection to a remote server
- bandit0@bandit.labs.overthewire.org specifies the username and the server to connect to
- The -p 2220 option tells SSH to connect to port 2220

## 2. Level 0→1

Goal: Find the password for the next level from a file named “readme”

Details:

- Server: bandit.labs.overthewire.org
- Port: 2220
- Username: bandit1
- Password: bandit1

Commands used:

```
ssh bandit0@bandit.labs.overthewire.org -p 2220
```

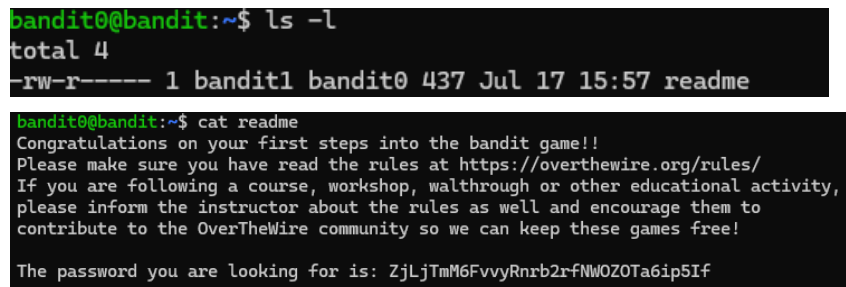
**List files in home directory**

```
ls -l
```

**Read the file**

```
cat readme
```

Screenshot:



```
bandit0@bandit:~$ ls -l
total 4
-rw-r----- 1 bandit1 bandit0 437 Jul 17 15:57 readme

bandit0@bandit:~$ cat readme
Congratulations on your first steps into the bandit game!!
Please make sure you have read the rules at https://overthewire.org/rules/
If you are following a course, workshop, walkthrough or other educational activity,
please inform the instructor about the rules as well and encourage them to
contribute to the OverTheWire community so we can keep these games free!

The password you are looking for is: ZjLjTmM6FvvyRnrb2rfNWOZOTa6ip5If
```

Brief Explanation:

- The “ls -l” command is used to list files and directories in long format
- The “cat readme” command is used to display the contents of the file named readme
- We obtain the password needed for the next level:  
**ZjLjTmM6FvvyRnrb2rfNWOZOTa6ip5If**

### 3. Level 1→2

Goal: Find the password for the next level, which is stored in a file called “\_”

Details:

- Server: bandit.labs.overthewire.org
- Port: 2220
- Username: bandit1
- Password: ZjLjTmM6FvvyRnrb2rfNWOZOTa6ip5If

Commands used:

```
ssh bandit1@bandit.labs.overthewire.org -p 2220
```

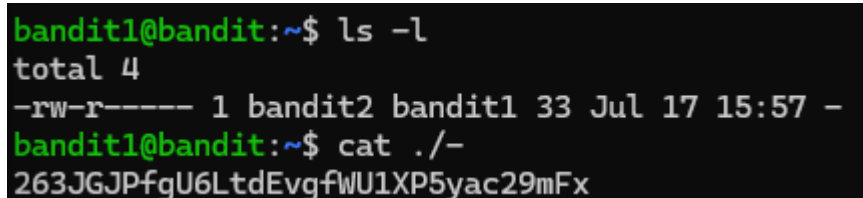
**List files in home directory**

```
ls -l
```

**Read the file**

```
Cat ./-
```

Screenshot:



```
bandit1@bandit:~$ ls -l
total 4
-rw-r----- 1 bandit2 bandit1 33 Jul 17 15:57 -
bandit1@bandit:~$ cat ./-
263JGJPfgU6LtdEvgfWU1XP5yac29mFx
```

Brief Explanation:

- The “ls -l” command is used to list files and directories in long format
- The “cat ./-” command is used to display the contents of the file named -
- We obtain the password needed for the next level:  
**263JGJPfgU6LtdEvgfWU1XP5yac29mFx**

#### 4. Level 2→3

Goal: Find the password for the next level, which is stored in a file called “spaces in this filename”

Details:

- Server: bandit.labs.overthewire.org
- Port: 2220
- Username: bandit2
- Password: 263JGJPfgU6LtdEvgfWU1XP5yac29mFx

Commands used:

```
ssh bandit2@bandit.labs.overthewire.org -p 2220
```

**List files in home directory**

```
ls -l
```

**Read the file** (Because the file contains spaces, we need to handle the filename with quotation marks)

```
Cat “spaces in this filename”
```

Screenshot:

```
bandit2@bandit:~$ ls -l
total 4
-rw-r----- 1 bandit bandit 33 Jul 17 15:57 spaces in this filename
```

```
bandit2@bandit:~$ cat "spaces in this filename"
MNk8KNH3Usiio41PRUEoDFPqfxLPISmx
```

Brief Explanation:

- The “ls -l” command is used to list files and directories in long format
- The “cat ./-” command is used to display the contents of the file named **spaces in this filename**
- We obtain the password needed for the next level:  
**MNk8KNH3Usiio41PRUEoDFPqfxLPISmx**

## 5. Level 3→4

Goal: Locate the password for the next file stored in a hidden file in the inhere directory

Details:

- Server: bandit.labs.overthewire.org
- Port: 2220
- Username: bandit3
- Password: MNk8KNH3Usiio41PRUEoDFPqfxLPISmx

Commands used:

```
ssh bandit3@bandit.labs.overthewire.org -p 2220
```

**Change current directory to “inhere”**

```
cd inhere
```

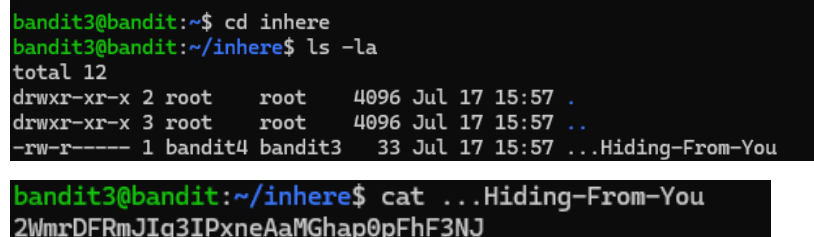
**List files including hidden ones**

```
ls -la
```

**Read the hidden file**

```
cat ...Hiding-From-You
```

Screenshot:



```
bandit3@bandit:~$ cd inhere
bandit3@bandit:~/inhere$ ls -la
total 12
drwxr-xr-x 2 root  root  4096 Jul 17 15:57 .
drwxr-xr-x 3 root  root  4096 Jul 17 15:57 ..
-rw-r----- 1 bandit4 bandit3  33 Jul 17 15:57 ...Hiding-From-You

bandit3@bandit:~/inhere$ cat ...Hiding-From-You
2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ
```

Brief Explanation:

- The “cd inhere” command changes the directory to inhere, where the hidden file is located
- The “ls -la” command lists all files in the directory, including hidden files
- “Cat ...Hiding-From-You” is used to read the hidden file we have found
- We obtain the password needed for the next level:  
**2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ**

## 6. Level 4→5

Goal: Find the password for the next level, which is stored in the only human-readable file in the inhere directory

Details:

- Server: bandit.labs.overthewire.org
- Port: 2220
- Username: bandit4
- Password: 2WmrDFRmJlq3IPxneAaMGhap0pFhF3NJ

Commands used:

```
ssh bandit4 @bandit.labs.overthewire.org -p 2220
```

**Change current directory to “inhere”**

```
cd inhere
```

**List files in inhere directory**

```
ls -l
```

**Identifying the file types**

```
file ./*
```

**Read the human readable file**

```
cat ./-file07
```

Screenshot:

```
bandit4@bandit:~$ cd inhere
```

```
bandit4@bandit:~/inhere$ ls -l
total 40
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file00
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file01
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file02
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file03
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file04
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file05
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file06
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file07
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file08
-rw-r----- 1 bandit5 bandit4 33 Jul 17 15:57 -file09
```

```
bandit4@bandit:~/inhere$ file ./*
./-file00: data
./-file01: data
./-file02: data
./-file03: data
./-file04: data
./-file05: data
./-file06: data
./-file07: ASCII text
./-file08: data
./-file09: data
```

```
bandit4@bandit:~/inhere$ cat ./-file07
4oQYVPkxZ00E005pTw81FB8j8lxXGUQw
```

Brief Explanation:

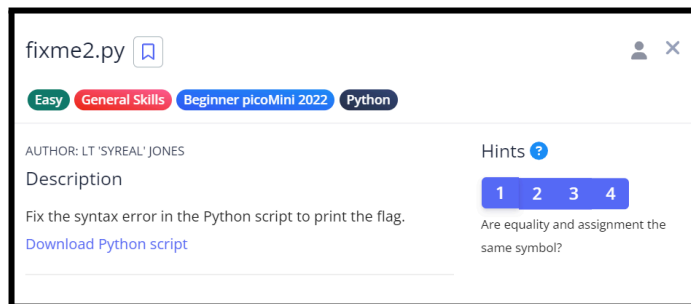
- The “cd inhere” command changes the directory to inhere, where the only human-readable file is located
- The “ls -l” command is used to list files and directories in long format
- “file ./\*” is used to identify the file type of each file in the directory, this helps find the file that is human-readable, as other files might be binary or otherwise unreadable
- The cat -file07 command reads the content of the file named -file07, which is identified as ASCII text (In an ASCII file, each alphabetic, numeric, or special character is represented with a 7 number)
- Finally we obtain the password for the next level (5→6),  
**4oQYVPkxZOOEOO5pTW81FB8j8lxXGUQw**



## 5 Random Pico-CTF Problems

### 1. fixme2.py

#### Challenge overview:



Goal: Fix the syntax error in the Python script to print the flag

Here's how the original script looks:

```
import random

def str_xor(secret, key):
    #extend key to secret length
    new_key = key
    i = 0
    while len(new_key) < len(secret):
        new_key = new_key + key[i]
        i = (i + 1) % len(key)
    return "".join([chr(ord(secret_c) ^ ord(new_key_c)) for (secret_c,new_key_c) in
zip(secret,new_key)])

flag_enc = chr(0x15) + chr(0x07) + chr(0x08) + chr(0x06) + chr(0x27) + chr(0x21) +
chr(0x23) + chr(0x15) + chr(0x58) + chr(0x18) + chr(0x11) + chr(0x41) + chr(0x09) +
chr(0x5f) + chr(0x1f) + chr(0x10) + chr(0x3b) + chr(0x1b) + chr(0x55) + chr(0x1a) +
chr(0x34) + chr(0x5d) + chr(0x51) + chr(0x40) + chr(0x54) + chr(0x09) + chr(0x05) +
chr(0x04) + chr(0x57) + chr(0x1b) + chr(0x11) + chr(0x31) + chr(0x5f) + chr(0x51) +
chr(0x52) + chr(0x46) + chr(0x00) + chr(0x5f) + chr(0x5a) + chr(0x0b) + chr(0x19)

flag = str_xor(flag_enc, 'enkidu')

# Check that flag is not empty
if flag = "":
    print('String XOR encountered a problem, quitting.')
else:
    print('That is correct! Here\'s your flag: ' + flag)
```

If we run the script above, we will get an error saying:

```
File "c:\Users\farhan\Documents\Personal Projects\Python\fixme2.py", line 22
    if flag = "":
        ^^^^^^^^^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

**Approach:**

To solve this problem, we need to fix the syntax error in the code

```
if flag = "": to → if flag == "":
```

After fixing the syntax error, we will get the decrypted flag

That is correct! Here's your flag:

picoCTF{3qu4l1ty\_n0t\_4551gnm3nt\_4863e11b}

**Additional Information:**

In the provided code, the flag (flag\_enc) is encrypted using XOR. XOR is a logical operation that compares two input bits and produces a single output bit. In Python, XOR is represented by the ^ operator. To decrypt the flag, the code applies XOR again with the same key ('enkidu'), effectively reversing the encryption process.

## 2. convertme.py

### Challenge overview:

convertme.py

Easy

General Skills

Beginner picoMini 2022

base

Python

AUTHOR: LT 'SYREAL' JONES

Description

Run the Python script and convert the given number from decimal to binary to get the flag.

[Download Python script](#)

Hints ?

1 2 3 4 5 6

Look up a decimal to binary number conversion app on the web or use your computer's calculator!

Goal: Run the Python script and convert the given number from decimal to binary to get the flag.

Here's how the script given looks like:

```
import random

def str_xor(secret, key):
    #extend key to secret length
    new_key = key
    i = 0
    while len(new_key) < len(secret):
        new_key = new_key + key[i]
        i = (i + 1) % len(key)
    return "".join([chr(ord(secret_c) ^ ord(new_key_c)) for (secret_c,new_key_c) in zip(secret,new_key)])

flag_enc = chr(0x15) + chr(0x07) + chr(0x08) + chr(0x06) + chr(0x27) + chr(0x21) + chr(0x23) + chr(0x15) + chr(0x5f) + chr(0x05) + chr(0x08) + chr(0x2a) + chr(0x1c) + chr(0x5e) + chr(0x1e) + chr(0x1b) + chr(0x3b) + chr(0x17) + chr(0x51) + chr(0x5b) + chr(0x58) + chr(0x5c) + chr(0x3b) + chr(0x42) + chr(0x57) + chr(0x5c) + chr(0x0d) + chr(0x5f) + chr(0x06) + chr(0x46) + chr(0x5c) + chr(0x13)

num = random.choice(range(10,101))

print('If ' + str(num) + ' is in decimal base, what is it in binary base?')

ans = input('Answer: ')

try:
    ans_num = int(ans, base=2)

    if ans_num == num:
        flag = str_xor(flag_enc, 'enkidu')
        print('That is correct! Here\'s your flag: ' + flag)
    else:
        print(str(ans_num) + ' and ' + str(num) + ' are not equal.')

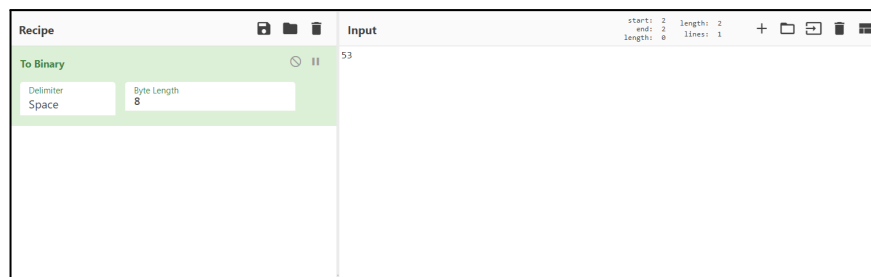
except ValueError:
    print('That isn\'t a binary number. Binary numbers contain only 1\'s and 0\'s')
```

The script generates a random decimal number between 10 and 100, here's the output after I run the script:

```
PS C:\Users\farhan> & C:/Users/farhan/AppData/Local/Programs/Python/Python312/python.exe  
If 53 is in decimal base, what is it in binary base?
```

### Approach:

It is asking to convert the number 53 from decimal base to binary. To convert it, I will use Cyberchef → by using the recipe “To Binary”



I will get the output

**Output** ✖ ✖ ✖

00110101 00110011

Hence, the binary equivalent of 53 is 110101. Then I will input the answer back to the terminal, and it will output the flag.

```
If 53 is in decimal base, what is it in binary base?  
Answer: 110101  
That is correct! Here's your flag: picoCTF{4ll_y0ur_b4535_722f6b39}
```

That is correct! Here's your flag: picoCTF{4ll\_y0ur\_b4535\_722f6b39}

### 3. unpackme.py

#### Challenge overview:

unpackme.py 

Medium Reverse Engineering picoCTF 2022 packing

AUTHOR: LT 'SYREAL' JONES

Hints ?

Description

(None)

Can you get the flag?

Reverse engineer this [Python program](#).

Goal: We need to reverse engineer this program →

```
import base64
from cryptography.fernet import Fernet

payload =
b'gAAAAABkzWGO_8MlYpNM0n0o718LL-w9m3rzXvCMRFghMRl6CSZwRD5DJOvN
_jc8TFHmHmfiiI8HWSu49MyoYKvb5mOGm_Jn4kkhC5fuRiGgmwEpxjh0z72dpi6
TaPO2TorksAd2bNLemfTaYPf9qiTn_z9mvCQYV9cFKK9m1SqCSr4qDwHXgkQpm
7IJAmtEJqyVUfteFLszyxv5-KXJin5BWf9aDPIskp4AztjsBH1_q9e5FIwIq48
H7AaHmR8bdvjcw_ZrvhAIOInm1oM-8DjamKvhh7u3-lA=='

key_str = 'correctstaplecorrectstaplecorrec'
key_base64 = base64.b64encode(key_str.encode())
f = Fernet(key_base64)
plain = f.decrypt(payload)
exec(plain.decode())
```

This python program is used to decrypt an encrypted message. But when I run this program, it will give this output:

```
PS C:\Users\farhan> & C:/Users/farhan/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe "c:/Users/farhan/Documents/Personal Projects/Python/Projects/3.unpackme.py"
What's the password?
```

### Approach:

Just running the program does not give us the flag that we need because the script is designed to prompt for a password. By adding `print(plain.decode())`, it will reveal the decrypted Python script's content, which includes the password check and the flag.

```
import base64
from cryptography.fernet import Fernet

payload =
b'gAAAAABkzWGO_8MlYpNM0n0o7l8LL-w9m3rzXvCMRFghMRl6CSZwRD5DJOvN
_jc8TFHmHmfii8HWSu49MyoYKvb5mOGm_Jn4kkhC5fuRiGgmwEpxjh0z72dpi6
TaPO2TorksAd2bNLemfTaYPf9qiTn_z9mvCQYV9cFKK9m1SqCSr4qDwHXgkQpm
7IJAmtEJqyVUfteFLszyxv5-KXJin5BWf9aDPIskp4AztjsBH1_q9e5FIwIq48
H7AaHmR8bdvjCW_ZrvhAIOInm1oM-8DjamKvhh7u3-lA=='

key_str = 'correctstaplecorrectstaplecorrec'
key_base64 = base64.b64encode(key_str.encode())
f = Fernet(key_base64)
plain = f.decrypt(payload)
print(plain.decode()) #<--- reveal the decrypted Python
script's content
exec(plain.decode())
```

Running this program now will show the decrypted script's content

```
pw = input('What\'s the password? ')

if pw == 'batteryhorse':
    print('picoCTF{175_chr157m45_5274ff21}')
else:
    print('That password is incorrect.')
```


```
What's the password? batteryhorse
picoCTF{175_chr157m45_5274ff21}
```



It will then also prompt the user for a password which is `*batteryhorse*`, once inputted the Flag will be obtained:

`picoCTF{175_chr157m45_5274ff21}`

#### 4. interendec

##### Challenge overview:

interendec 

Easy Cryptography picoCTF 2024 base64 browser\_webshell\_solvable caesar

AUTHOR: NGIRIMANA SCHADRACK

### Description

Can you get the real meaning from this file.  
Download the file [here](#).

### Hints

1

Engaging in various decoding processes is of utmost importance

Goal: Can you get the real meaning from this file? Download the file here.





Once we have downloaded the file and opened it, we will get this line of string →




```
File Edit View
YidkM0JxZGtwQlRYdHFhR3g2YUhsZmF6TnFlVGwzWVROclgya3lNRFJvYTJvMmZRPT0nCG==
```

##### Approach:

I used cyberchef to decode the string from Base64, here's how the result looks:

### Recipe





**From Base64**   

Alphabet  
A~Za~z0~9+~=



☒ Remove non-alphabet chars




☐ Strict mode

### Input

YidkM0JxZGtwQlRYdHFhR3g2YUhsZmF6TnFlVGwzWVROclgya3lNRFJvYTJvMmZRPT0nCG==

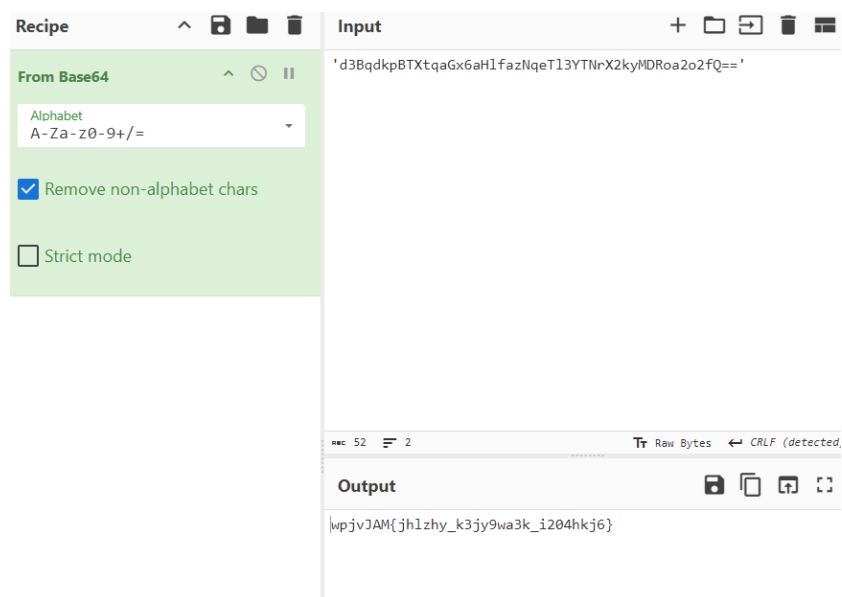
74 2  Raw Bytes 

**Output**   

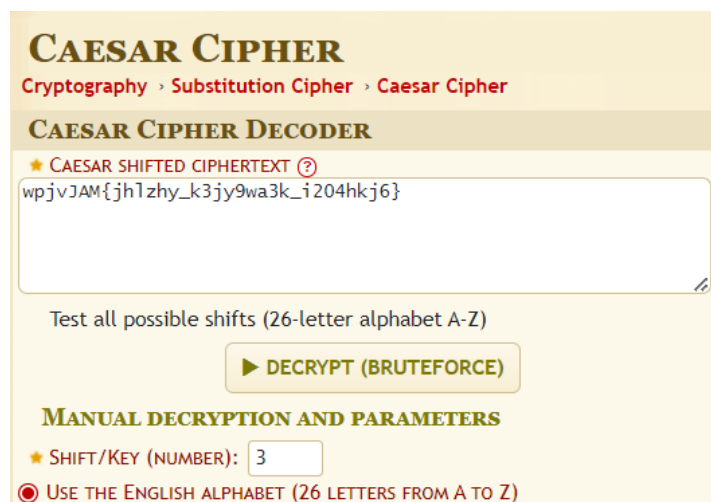
```
b'd3BqdkpBTXtqaGx6aH1fazNqeTl3YTNrX2ky#DRoa2oZfQ=='
```

The result was a string:

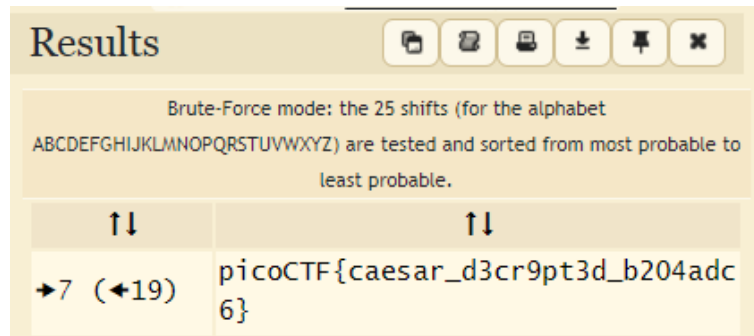
`b'd3BqdkpBTXtqaGx6aHlfazNqeTl3YTNrX2kyMDRoa2o2fQ=='`, which is still Base64 encoded. After removing the `b` (byte object) prefix and decoding the string again using CyberChef, the result was:



The decoded string seemed to be Caesar-encrypted. Using a Caesar Cipher brute-force tool, I decrypted it using <https://www.dcode.fr/caesar-cipher>, to reveal the final flag:







Hence the Flag is `picoCTF{caesar_d3cr9pt3d_b204adc6}`

### Additional Information:

The Caesar cipher is a simple encryption technique that was used by Julius Caesar to send secret messages to his allies. It works by shifting the letters in the plaintext message by a certain number of positions, known as the “shift” or “key”.

It’s simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on.

Source: <https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/>

## 5. Unminify

### Challenge overview:

Unminify 



Easy

Web Exploitation

picoCTF 2024

obfuscation

browser\_webshell\_solvable

minification

AUTHOR: JEFFERY JOHN

### Description

I don't like scrolling down to read the code of my website, so I've squished it. As a bonus, my pages load faster!  
Browse [here](#), and find the flag!

This challenge launches an instance on demand.

Its current status is: **RUNNING**

Instance Time Remaining: **28:07**

**Restart Instance**

Hints 

1 2 3

**Goal:** Find the flag contained in the webpage

### Approach:

This problem is fairly a simple one, here's how the webpage looks:



By using inspect element and scrolling down, the flag can be easily spotted →

```
  class="picoctf{}">
  class="picoctf{}">
  v class="picoctf{}" style="padding-top:30px;border-radius:3%;box-shadow:0 5px 10px #0000004d;width:50%;align
  
  <div class="picoctf{}" style="width:85%">
    <h2 class="picoctf{}">Welcome to my flag distribution website!</h2>
    <div class="picoctf{}" style="width:70%">
      <p class="picoctf{}">If you're reading this, your browser has succesfully received the flag.</p>
      <p class="picoCTF{pr3tty_c0d3_743d0f9b}"></p>
      <p class="picoctf{}">I just deliver flags, I don't know how to read them...</p>
    </div>
  </div>
  <br class="picoctf{}">
  </div>
```

The flag is the line of code I have highlighted above.

Hence the flag is `picoCTF{pr3tty_c0d3_743d0f9b}`