Write-up

Shellmates Mini-CTF 2018 – Guess the token 3

Amina BALI
SHELLMATES MEMBER
ea_bali@esi.dz

Thanks

Special thanks to KIMOUCHE Mohamed and BOUTHIBA Abderraouf who organized this Mini-CTF and for all the help they gave me and what I learnt from them.

And of course, thanks to all Shellmates members (ntouma haylin 😉).

Challenge description

Title: Guess the token 3

Category: Web

Description:

Cette fois-ci, Jack pense que son code est infaillible ! Prouvez-lui qu'il a tort !

This time, Jack thinks his code is infallible! Prove him he's wrong!

http://192.168.0.200/guess_the_token_3/index.php

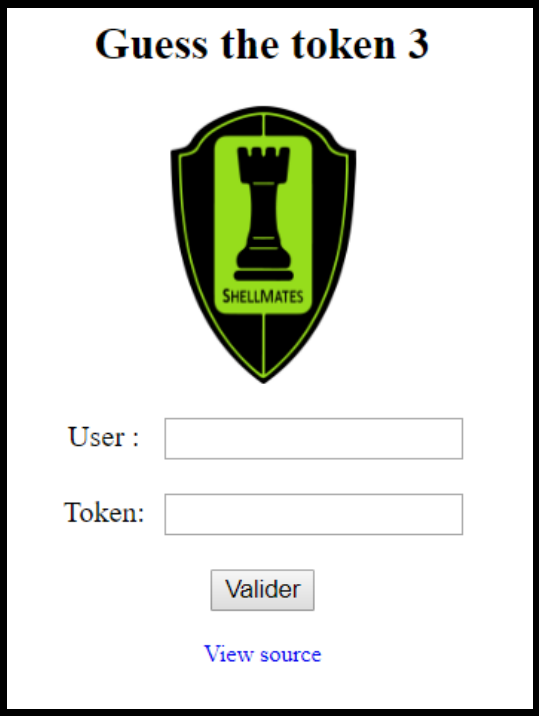
Points: 100

Difficulty: Easy to medium

Author: Raouf ou Mohamed ?

Analysis

When we click on the link in the description this simple page shows up:



The ‘view source’ catches our attention, here is the php source that we get:

```php
<?php
require_once "config.php";
if (!empty($_POST)){
  $username = $_POST["username"];
  $p_token = $_POST["token"];
  //$current_time = time();
  //$TOKEN = $current_time.rand(1,50);
  if ($p_token===$TOKEN && $username.$TOKEN.rand(1,31337)=="0e830400451993494058024219903391")
    $msg="<font color=green> Well done here is your gift: </font> $FLAG <br><br>";
  else
    $msg="<font color=red> Wrong token </font> <br><br>";
}
?>

<link rel="stylesheet" href="style.css" />
<h2>Guess the token 3</h2>
<img src="shellmates.png" width=100 height=150 /><br><br>
<center>
<form method="post" action="#">
  <div class="row"> <label> User : </label> <input name="username" type="text"/>  <br><br> </div>
  <div class="row"> <label> Token: </label> <input name="token" type="text"/><br><br> </div>
  <input name="submit" type="submit" />
</form>
</center>
<?php echo $msg; ?>
<a href="source.php"> View source </a>
```

So as we can see, the token is very simple to generate, all we do is get the current timestamp with the php function time() and concatenate it to a random number in the range [1, 50] (included). We got this from the comment section in our source!

Now, let's move to the condition part! We first check if the token given is correct by comparing it strictly to the stored token in $TOKEN. (Notice the "===" which stands for strict comparison, means that type and value have to match in both sides!)

Now, we move to the second part of the comparison:

$username.$TOKEN.rand(1,31337)=="0e830400451993494058024219903391"

So, the username concatenated with the token value and a random number between 1 and 31337 has to match with "0e830400451993494058024219903391". Not that easy to guess the username, right? BUT, when we focus more on this part, we notice the use of "==" which stands for loose comparison, means that only values have to match after determining the type automatically by php based on the context (type juggling)! This is a great way to go!

So, what we should do is provide a username that will make php think both sides are equal without being really equal!!! Means that even if the input does not form the right string, php will consider the condition as verified! How can we do this?

Well, if we focus more on the string "0e830400451993494058024219903391", we see that it starts with 0e! this makes us think about exponent numbers in $0e^x$ format which gives the value of 0 when evaluating! Okey, good! This leads us think of providing an input that will make php also think it's an exponent number as we did! For this, we observe the left part of the loose comparison: $username.$TOKEN.rand(1,31337), we notice that if we provide a username with value of 0e, we will get the same format as the previous string! (knowing that the token is a sequence of numbers!). So, we got it! We just need to make username equal to 0e and php will consider it as a numerical number and evaluate it to 0 on both sides!

Resolution

Now that we analyzed and understand how the whole thing works, we only need to write a script that will put “0e” as the username value and generate all the possible tokens and try them one by one until the flag is returned.

As we practically have the code that generates the tokens we will reuse it as follow:

```php
<?php

    $current_time = time();
     for ($x = 1; $x <= 50; $x++)
     {
         $TOKEN = $current_time.$x;
         echo $TOKEN . "\xa";
     }
?>
```

We put the generated tokens into a text file (tokens.txt) that we will use in our python script to post the tokens and “0e” as username:

```python
import requests
import re

file = open("tokens.txt", "r").readlines()

for line in file:
    line = line.strip('\n')
    r = requests.post("http://192.168.0.200/guess_the_token_3/index.php", data={'username': '0e', 'token': line})
    if not re.search('Wrong token', r.text):
        print ("flag : " + r.text)
```

When we execute the script (python ./guess_the_token_3.py) we finally get the flag: Shellmates{php_l00sy_0peration_is_n0t_a_g00d_0ption}

PS. Of course, we could have put everything in the same script: generate the token and test them! I used the php code to generate the tokens just because the code was already there, so I reused it directly!

What we learn from this task

Through this task we learned about comparison types in php which are (according to php comparison operators manual):

1. Loose comparison: (like == and !=) also called "equal", for example $a==$b, TRUE if $a is equal to $b after type juggling ! You may ask ~~wth~~ is type juggling, so, type juggle in php is simply the fact that when we declare a variable, specifying its type is not required, this one is determined by the value or the context of this variable. So, for example if we assign an integer value to a variable, it will automatically become an int and if we assign to it a string it will become a string!
   In our case, we had a great example of juggling type based on the context! As we saw, we provided "0e" as username and after php concatenates it with the token and a random number, it interpreted it as a numerical string, an exponent number actually, in the form of $0e^x$! So, it considered it as 0 (after conversion and calculation) and automatically the other part of the loose comparison too (0e830400451993494058024219903391), so, we ended up with both sides equal to zero and automatically a verified condition!

   For more details about php loose comparison, go check this very interesting article.

2. Strict comparison: (like === and !==) also called "identical", here types and values matter! Types will not change "magically", no conversion going on. for example, $a===$b, TRUE if $a is equal to $b, and they are of the same type.

Thanks for reading 😊