Ballin' Write-Up - ArkCon 19 Challenge

> Author: Doron Naim

> Value: 250 points

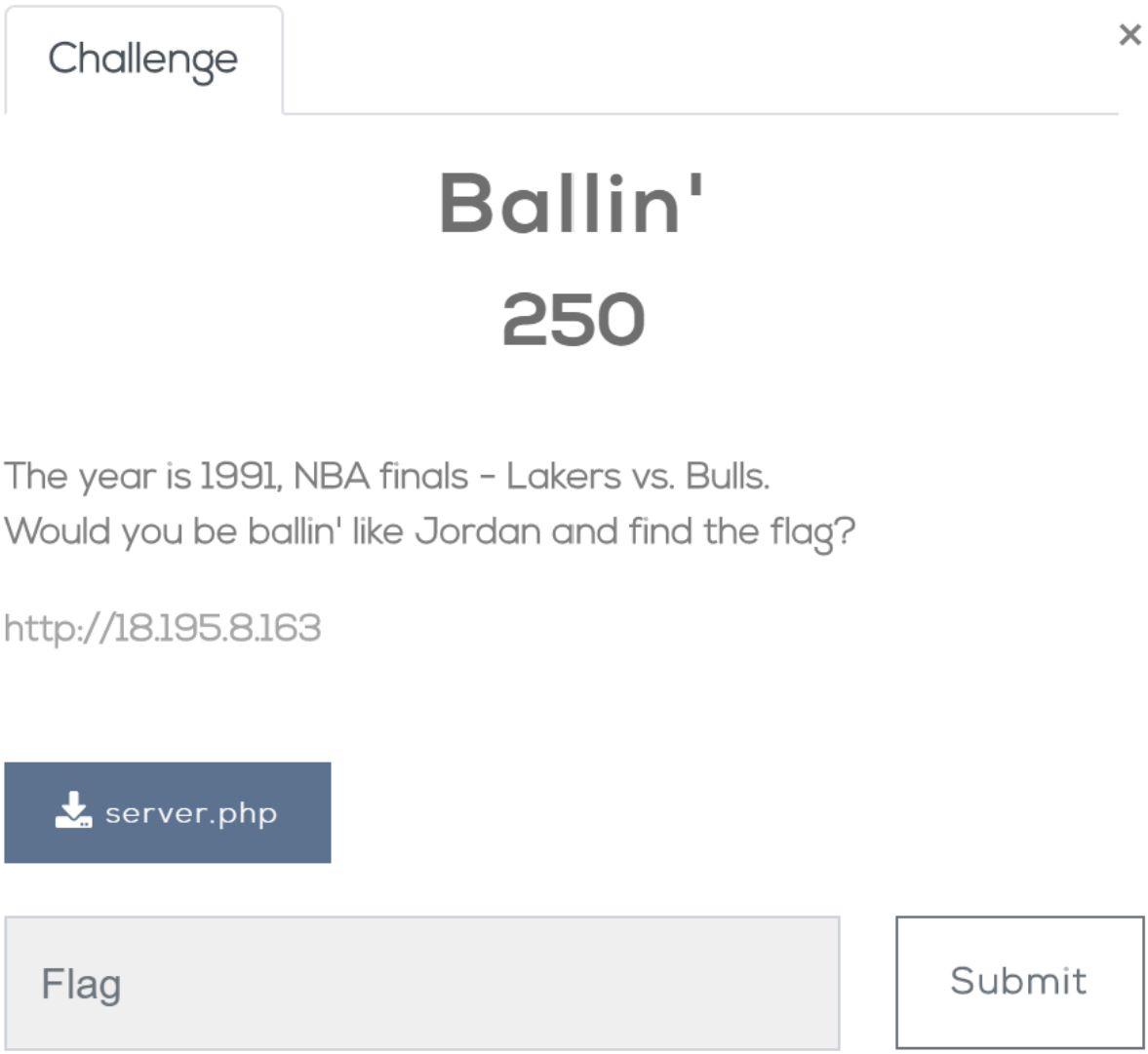
Background

Ballin' is an HTTP server challenge written in PHP.

The challenge starts with a given rotated PHP server – the solvers need to find the needed rotation and later focus on the server itself.

Their goal is to bypass all conditions send to EXIT and print the flag stored on the server.

Note: There is more than one solution. You can check out the solution published on Digital Whisper.



Solution

1. Code Resolving

Once downloading the *server.php* file, you get the following content:

```
zohwncih r0l($mnlcha, $guacw_eys) { #bnnjm://mnuweipylzfiq.wig/koymncihm/14673551/yhwlsjn-xywlsjn-qcnb-ril-ch-jbj } $zfua = "UleWih{?}"; $fcmn = ullus('wlw32', 'gx5', 'mbu1'); $ufai = $fcmn[ullus_luhx($fcmn)]; cz (!ygjns($_JIMN['eys']) || !ygjns($_JIMN['bulugy'])){ $vumeyn = bumb($ufai, r0l($zfua, $_JIMN['eys'])); $bulugy = bumb($ufai, mnllyp($_JIMN['bulugy'])); } yfmy{ ywbi ("Jfyumy wfimyx nby xiil vybchx sio\h"); yrcn; } $wixy = "20190429_71070_y7707_1312_3_14159265359"; cz (!ygjns($_JIMN['wiuwb']) || !ygjns($_JIMN['dylmys'])) $vumeyn = mnl_lyjfuwy("_", $_JIMN['wiuwb'], movmnl($wixy, $_JIMN['dylmys']).movmnl($wixy, 0, $_JIMN['dylmys'])); ywbi ("BULUGY!\h"); $vuff = bumb('gx5', $vumeyn); cz (cmmyn($_JIMN['vuff']) && cmmyn($_JIMN['dylmys']) ){ $vuff = movmnl(r0l(juwe("B*", $_JIMN['vuff']), $vuff), -8) * $_JIMN['dylmys']; $bulugy = bumb($ufai, r0l($bulugy, $vumeyn)); } cz( $vumeyn != bumb($ufai, $vuff)) yrcn; $eys = bumb($ufai, $bulugy); ywbi ("U bchn:" . mnl_mbozzfy($eys . $zfua) . "\h"); cz ($eys == $_JIMN['eys']) ywbi ("Sio uly nby AIUN: " . $zfua); yrcn;
```

obviously, this is not the code we wished for. You can assume there is a small trick you should do in order to make it clear. By a basic brute force or a basic examination of the strings, you'll be able to find the way to make it readable.

But let's take the full path :)

Once connecting to the server using the web-browser, we get randomized outputs by the server. One of them is: *"People say Jordan went through a purple PATCH, maybe you should too"*.

As you can see, in capital letters we have the word PATCH, since the challenge is an HTTP challenge and deals with POST and GET requests, you might think you should try a PATCH request as well.

After sending a PATCH request, you'll get the following string: *"Healthy rotation is a **rotation** with a Magic's number.. keep in mind we play ball"*.

Having that, we understand there is a rotation to be made. Keeping in mind the description of the challenge, which is: "*The year is 1991, NBA finals - Lakers vs. Bulls*" - so we can look for the words "Magic" and NBA/Lakers/Chicago and we'll find that Magic is actually a player (Magic Johnson), and his jersey number is 32.

After completing a rotation of 32 (or 6=32-26) - you get the following PHP server code:

```php
<?php

function xOr($string, $magic_key) {
  #https://stackoverflow.com/questions/14673551/encrypt-decrypt-with-xor-in-php
}


$flag = "ArkCon{?}";
$list = array('crc32', 'md5', 'sha1');
$algo = $list[array_rand($list)];

if (!empty($_POST['key']) || !empty($_POST['harame'])){

    $basket = hash($algo, xOr($flag, $_POST['key']));
    $harame = hash($algo, strrev($_POST['harame']));
}
else{
    echo ("Please closed the door behind you\n");
```

```
    exit;
}

$code = "20190429_71070_e7707_1312_3_14159265359";


if (!empty($_POST['coach']) || !empty($_POST['jersey']))
    $basket = str_replace("_", $_POST['coach'], substr($code,
$_POST['jersey']).substr($code, 0, $_POST['jersey']));

echo ("HARAME!\n");

$ball = hash('md5', $basket);

if (isset($_POST['ball']) && isset($_POST['jersey']) ){
    $ball = substr(x0r(pack("H*", $_POST['ball']), $ball), -8) * $_POST['jersey'];
    $harame = hash($algo, x0r($harame, $basket));

}


if( $basket != hash($algo, $ball))
    exit;

$key = hash($algo, $harame);
echo ("A hint:" . str_shuffle($key . $flag) . "\n");

if ($key == $_POST['key'])
    echo ("You are the GOAT: " . $flag);

exit;

?>
```

Now we can start.

2. Understanding the code

The code has few parts:

a. X0r function

We don't get the function itself but a reference to it on stackoverflow.

```
function x0r($string, $magic_key) {
  #https://stackoverflow.com/questions/14673551/encrypt-decrypt-with-xor-in-php
}
```

b. Variables and Parameters

In the beginning, we have 3 variables:

```
$flag = "ArkCon{?}";
$list = array('crc32', 'md5', 'sha1');
$algo = $list[array_rand($list)];
```

As we can see, the $flag is unknown and stored on the server. The $algo is randomly being chosen every time we make an HTTP request, can be either 'crc32'/'md5'/'sha1'.

So even if you have the winning input for that challenge - sometime it won't work due to the algorithm you chose to solve by.

More variables are: $code, $ball, $harame, $basket and $key.

In order to solve the challenge, one needs to send a POST request with the following five parameters:

'key', 'harame', 'coach', 'jersey' and 'ball'.

c. Magic Hash

Another variable, which is critical for the solution of this challenge, is the $code variable:

```
$code = "20190429_71070_e7707_1312_3_14159265359";
```

The ones who played with PHP in the past, maybe know that there are hashes called Magic Hashes, and they start with '0e'/'00e' and followed by numbers. Later on, we'll see how the code is performing a manipulation on this string so you can turn this $code to a valid Magic Hash.

What are Magic Hashes?

Let's see the following case:

```
php > var_dump("240610708" == "QNKCDZO");
bool(false)
php > var_dump(md5("240610708") == md5("QNKCDZO"));
bool(true)
php > echo md5("240610708");
0e462097431906509019562988736854
php > echo md5("QNKCDZO");
0e830400451993494058024219903391
```

At the first line, when we compare two different string, we get **false**.

Next, we compare those two string again, but after calculating their md5 hash - and we get **true**.

This is really odd, later we printed the hashes and as we said before, they start with 'e0' and followed by numbers.

What the PHP engine is really doing is an implicit cast. It thinks it is a mathematical value so it does translate, for example, the **0e**46209743190650901956298873685 4 to **0*10^**46209743190650901956298873685 4.

So the 0e is being translated to a mathematical operation of 0*10^..... what makes the result of this phrase to be always 0. 0 always equals to 0.

PHP is not doing this translation when you use '===' instead of '=='.

d. Conditions

The challenge will be solved only if we'll be able to pass all conditions.

The first one is quite easy, you have to supply in your POST request two parameters: 'key' and 'harame'. Otherwise, you'll be kicked out of the server.

```
if (!empty($_POST['key']) || !empty($_POST['harame'])){

    $basket = hash($algo, xOr($flag, $_POST['key']));
    $harame = hash($algo, strrev($_POST['harame']));
}
else{
    echo ("Please closed the door behind you\n");
    exit;
}
```

As we can see, there are two variables that are being updated by this condition:

1. $basket - basically we don't care about that line since $basket is being overwritten later. Nevertheless, the 'key' value is important for the final condition.
2. $harame - this one is easy, you can guess according to the $algo you build your solution.

The second one is expecting to get another two parameters, 'coach' and 'jersey'. As it seems, the code within the condition is replacing the "_" character with 'coach' parameter value. Before doing that, the code is kind of rotating the string (ROL) according to the value stored in 'jersey' parameter.

This will help us to transform the string to a Magic Hash as explained above.

```
if (!empty($_POST['coach']) || !empty($_POST['jersey']))
    $basket = str_replace("_", $_POST['coach'], substr($code,
$_POST['jersey']).substr($code, 0, $_POST['jersey']));
```

Example of the behavior of that line:

```
php > echo str_replace("_", '', substr('abcdefg', 3).substr('abcdefg', 0, 3));
defgabc
php >
```

The third condition is taking the 'ball' and 'jersey' parameters and set values in $ball and $harame variables.

The $ball variable is getting the last 8 characters of the XOR result of 'ball' parameter and $ball variable which is known to us, then it doubles it by 'jersey' parameter (here we get the sense this should be a number stored in that parameter. Later on, we'll see the entire flow of the solution.

$harame is being set with the hash result of X0r($harame, $basket). Those variables' content is known to us.

```
if (isset($_POST['ball']) && isset($_POST['jersey']) ){
    $ball = substr(x0r(pack("H*", $_POST['ball']), $ball), -8) * $_POST['jersey'];
    $harame = hash($algo, x0r($harame, $basket));

}
```

The next condition is checking whether $basket is NOT equal to the hash result of the $ball variable. In case it is not equal, we'll be kicked out of the server. This is the condition the entire challenge is based on.

In this step, we'd like both $basket and the result of the hash calculation to be Magic Hashes OR equal to 0.

```
if( $basket != hash($algo, $ball))
    exit;
```

The last condition is the one who can give you the flag while checking if $key variable is equal to the 'key' parameter we sent on our POST request. Otherwise, we'll be kicked out of the server.

```
if ($key == $_POST['key'])
    echo ("You are the GOAT: " . $flag);
```

3. Crack it down

So now, after you have the knowledge needed for solving the challenge, we can continue to this exciting part of solving the challenge.

This is a packet's parameter (one of many, as being said - there is more than one packet which can solve the challenge) which can solve the challenge:

```
        {'key':'5ec1910c16640e72e6729f771bea368b',
        'harame': [],
        'jersey': '13',
        'ball': "040a0c085a02075403010d010f545304035b51060f075503080102085e065201",
        'coach': ''
        }
```

Now let's do the entire path together, explaining how and why we chose those parameters.

We know one of the algorithms has been set. We will build our solution based on 'md5' hashing (on Digital Whisper, you have an example of 'crc32' based solution).

```
$flag = "ArkCon{?}";
$list = array('crc32', 'md5', 'sha1');
$algo = $list[array_rand($list)];

if (!empty($_POST['key']) || !empty($_POST['harame'])){

    $basket = hash($algo, xOr($flag, $_POST['key']));
    $harame = hash($algo, strrev($_POST['harame']));
}
else{
    echo ("Please closed the door behind you\n");
    exit;
}
```

1. Starting at the first condition, as we said before we don't care about $basket, only $harame is interesting since it's being used later on.

   Here we chose to set '**[]**' as the parameter since we know the value of "strrev" function over '[]' is NULL.

   So, let compute a 'md5' hash of NULL:

   ```
   php > var_dump(strrev([]));
   PHP Warning:  strrev() expects parameter 1 to be string, array given in php shell code on line 1
   NULL
   php > echo md5(strrev([]));
   PHP Warning:  strrev() expects parameter 1 to be string, array given in php shell code on line 1
   d41d8cd98f00b204e9800998ecf8427e
   php > md5(NULL);
   php > echo md5(NULL);
   d41d8cd98f00b204e9800998ecf8427e
   ```

   For now, the value of **$harame is 'd41d8cd98f00b204e9800998ecf8427e'.**

   Let's move forward in the code:

```
$code = "20190429_71070_e7707_1312_3_14159265359";


if (!empty($_POST['coach']) || !empty($_POST['jersey']))
    $basket = str_replace("_", $_POST['coach'], substr($code,
$_POST['jersey']).substr($code, 0, $_POST['jersey']));

echo ("HARAME!\n");
```

2. As explained in detail above, we deal here with Magic Hash trick. We want to get rid of the '_' so we will set 'coach' parameter to be empty. In 'jersey' parameter we set the value 13. Let's see an example of those operation's result:

```
php > $code = "20190429_71070_e7707_1312_3_14159265359";
php > echo substr($code, 13).substr($code, 0, 13);
0_e7707_1312_3_1415926535920190429_7107
php > echo str_replace("_", "", "0_e7707_1312_3_1415926535920190429_7107");
0e770713123141592653592019042971 07
```

So now, **$basket = 0e770713123141592653592019042971 07**, like we wanted - now it's a valid Magic Hash as described before.

FYI - the original string of $code is assembled by the following ideas:

- 20190424 - it's the date of ArkCon conference
- 71070 - it's pimp in Hebrew :)
- e7707 - it's error
- 1312 - it's the known phrase "ACAB"
- 3_14159265359 - it's PI

Moving forward:

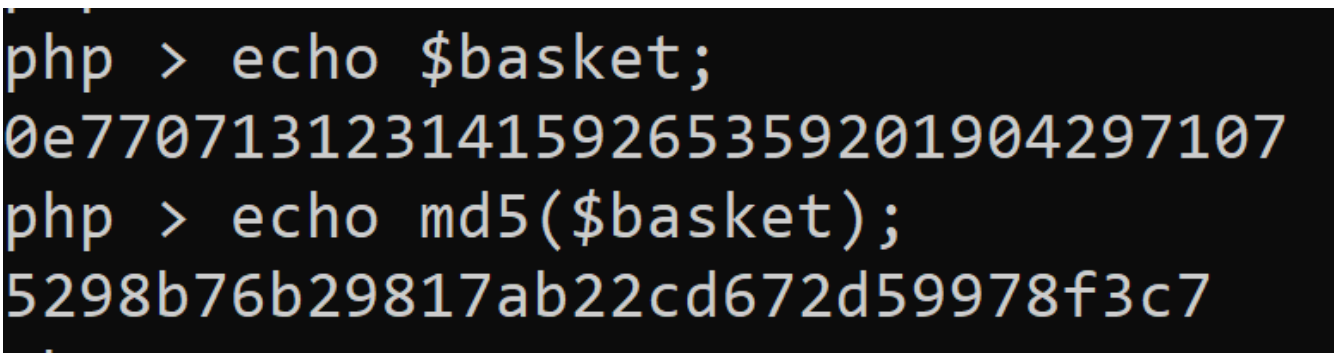
```php
$ball = hash('md5', $basket);

if (isset($_POST['ball']) && isset($_POST['jersey']) ){
    $ball = substr(x0r(pack("H*", $_POST['ball']), $ball), -8) * $_POST['jersey'];
    $harame = hash($algo, x0r($harame, $basket));
}

if( $basket != hash($algo, $ball))
    exit;
```

3. Easy to see that if we got so far, $ball variable is being set with the 'md5' hash result of $basket.

   Since we know the value of $basket, this is easy for us:



So now, **$ball= 5298b76b29817ab22cd672d59978f3c7**.

In order to get a sense of what should be 'ball' parameter (we assume 'jersey' parameter is 13, as it helped us with the Magic Hash creation), we need to see what use the code does with this $ball variable. Let's see the condition below it, we'd like to pass this condition - otherwise, we are out of the game. In order to do that, as mentioned before, we'd like both $basket and the hash calculation to be Magic hash OR 0. In case those are Magic Hashes, they will be translated to 0 and we'll jump over the exit command.

Based on the code, $ball needs to be a value in HEX, then the code converts the HEX to String and performs an XOR between that and $ball (which we have). Then, if we'll multiply it by 13 we should be getting a value that can turn to a Magic Hash. Sounds complicated, but it doesn't.

All we need is to find a value that by performing a 'md5', will turn to a Magic Hash, and check whether it can be divided by 13.

If you'll look for Magic Hashes on the internet, you'll find a couple of options. One of them is: '**240610708**', which can be divided by 13: 240610708/13 = 18508516.

So the left side of this command needs to be equal to 18508516:

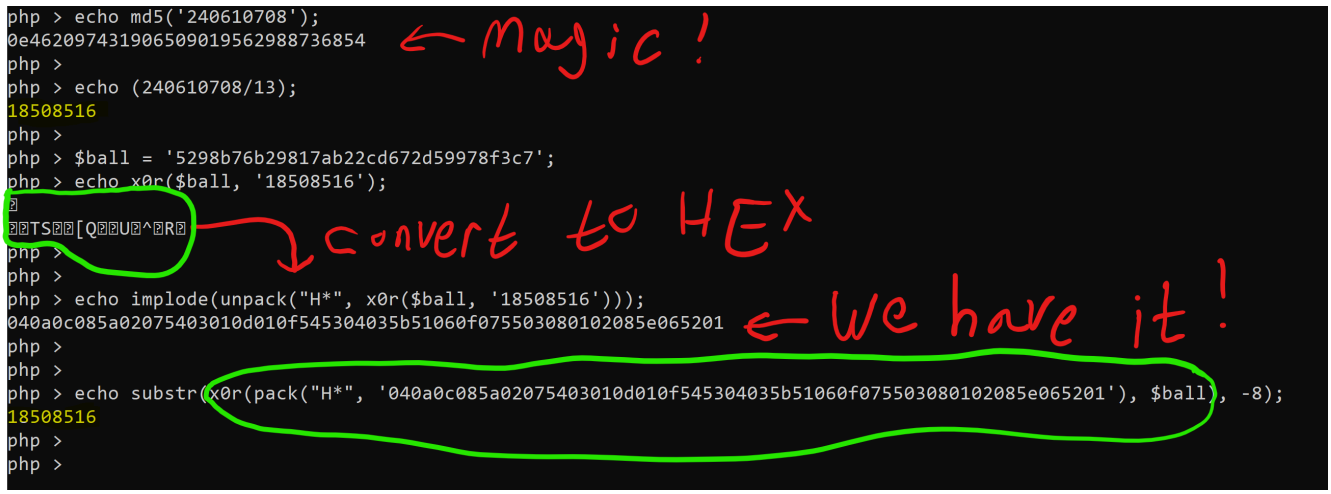
```
substr(x0r(pack("H*", $_POST['ball']), $ball), -8) <= 18508516
```

Seems still complicated, but let's move forward.

The fact that XOR is a symmetric function, allow us to understand what should we place in 'ball' parameter.

We know that $ball= 5298b76b29817ab22cd672d59978f3c7, so we need to XOR both '18508516' and '5298b76b29817ab22cd672d59978f3c7', and the result of it should be converted to HEX!

Let do it together:



[green = green]

So we have it, if we set 'ball' parameter to be '**040a0c085a02075403010d010f545304035b51060f075503080102085e065201**' it will be calculated to 18508516 and then it will be multiply by 13. Eventually we will get what we wished for: **$ball = 240610708**.

Now both **$basket = 0e770713123141592653592019042971 07** and **hash('md5', $ball = 240610708) = 0e462097431906509019562988736854** are Magic Hashes and we can overcome this vicious condition.

$harame is also being changed (we know $harame is 'd41d8cd98f00b204e9800998ecf8427e'), now after the XOR between $harame and $basket, we set a new value into **$harame = 4f2dd49f24955562b9422bb27c7b91de.**

```
php > $basket = '0e770713123141592653592019042971O7';
php > $harame = 'd41d8cd98f00b204e9800998ecf8427e';
php > $harame = md5(x0r($harame,$basket));
php > echo $harame;
4f2dd49f24955562b9422bb27c7b91de
```

Let's move on to the final condition:

```
$key = hash($algo, $harame);
echo ("A hint:" . str_shuffle($key . $flag) . "\n");

if ($key == $_POST['key'])
    echo ("You are the GOAT: " . $flag);

exit;
```

Now it's much easier.

The $key is being set with the 'md5' hash of $harame.

```
php > echo md5($harame);
5ec1910c16640e72e6729f771bea368b
```

$key = 5ec1910c16640e72e6729f771bea368b.

Well, this is clear, we'll set this value under the 'key' parameter in our POST request, and we're done!

(All underlined values are the final values of the request which needed to be made).

Again, those are the parameters we got:

```
        {'key':'5ec1910c16640e72e6729f771bea368b',
        'harame': [],
        'jersey': '13',
        'ball': "040a0c085a02075403010d010f545304035b51060f075503080102085e065201",
        'coach': ''
        }
```

The server's response:

```
[Running] python -u "c:\Users\[redacted]\Desktop\kaki.py"
HARAME!
A hint:241SD59_05RR1fT16ea7!3_RJeYeNH0b!0{3recv3927o3bBP_A13738InL_6T66C}4kc47_
You are the GOAT: ArkCon{J0RD4N_I5_TH3_B3ST_PL4Y3R_3v3R!!}

[Done] exited with code=0 in 0.507 seconds
```

The $flag is: `ArkCon{J0RD4N_I5_TH3_B3ST_PL4Y3R_3v3R!!}`