Circus

Problem Description:

On opening the link for the challenge, we are presented with following **PHP** code.

```php
<?php

if (isset($_GET['hash'])) {
    if ($_GET['hash'] === "10932435112") {
        die('r/woosh');
    }

    $hash = sha1($_GET['hash']);
    $target = sha1(10932435112);
    if($hash == $target) {
        include('flag.php');
        print $flag;
    } else {
        print "ehax{sad}";
    }
} else {
    show_source(__FILE__);
}

?>
```

The code checks for the **http query** "`hash`". If the hash query is set, it checks if the query equals "`10932435112`", in that case it exits with "`r/woosh`" on the screen. Next it uses **SHA1 hash** algorithm to hash the value of "`hash`" query and the number "`10932435112`". If both the hashes are equal, it gives the flag on the screen else gives "`ehax{sad}`" on the screen.

Basic keywords and Definitions:

PHP: PHP stands for "PHP: Hypertext Preprocessor". It is a general purpose programming language mainly used for web development.

HTTP Queries: A query string is a part of the URL of a webpage. It is used to assign values to specific parameters. Eg: https://example.com/page?name=val here parameter "`name`" is given the value "`val`".

Hashing: It is a function which is used to map data of arbitrary size to fixed size values. It is used for efficient indexing and enhancing security. It is difficult to reverse engineer usually.

SHA1: It is a hash function which takes an input and converts it to 160 bits or 20 bytes long output.

PHP Type Juggling: Before performing loose comparison in PHP, PHP does type conversions which is called type juggling.

Methodology:

So, first I tried using "?hash=010932435112", which gives "ehax{sad}". Even trying the hex representation of the number gave the same output ("?hash=0x28B9FB8A8"). Cryptographically, SHA1 is not considered to be secure as its has collisions have been found. I tried writing a small bash script which would try and find a hash collision for the number hoping to find some number small enough.

```
#!/bin/bash

sum=$(echo -n 10932435112 | sha1sum)

i=0
while true; do
  if [[ "$i" == "10932435112" ]]; then
    continue
  fi
  temp=$(echo -n "$i" | sha1sum)
  if [[ "$temp" == "$sum" ]]; then
    echo "$i"
    break
  fi
  ((i++))
done
```

But I wasn't able to find it because brute forcing this is computationally expensive.

On closer inspection of code, I found that "$hash" and "$target" were being compared using "==" which does loose comparison in PHP. Now we check if the hash of the number "10932435112" is something special. We can see the SHA1 hash of "10932435112" is "0e07766915004133176347055865026311692244". The hash starts with "0e" followed by only numbers. If a string hash starts with "0e" followed by only numbers, PHP interprets this as scientific notation and the hash is treated as a float in comparison operations.

By the rules of PHP **type juggling**, when comparing strings to numbers or other numeric strings ("1", "1e1", etc.), PHP converts those strings to numbers. So by the rules of type juggling, "0e12345" == "0" gives true in PHP. Therefore, "0e12345" == "0e6789" also evaluates to true.

Therefore, we need a string which is not "10932435112" that on computing the SHA1 hash of gives either 0s or "0e" followed only by numbers. These hashes are also sometimes called "**magic hashes**". On quick google search, the string "aarozm0k" gives SHA1 hash "0e66507019969427134894567494305185566735". More magic hashes can be found here: https://github.com/spaze/hashes. Now, adding "?hash=aarozm0k" to the challenge link in the end gives us the required flag.

ehax{l3mm3_jUggl3_dem_b4lls}