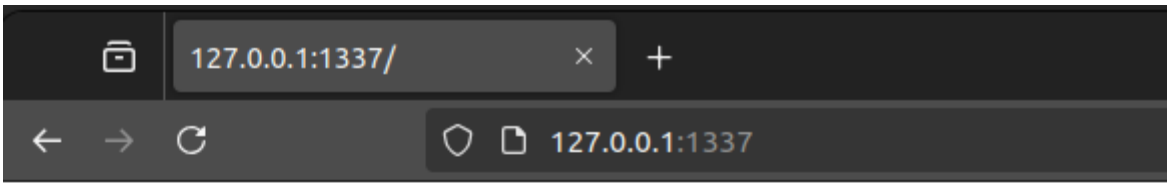
Circus Write-Up

As soon as I opened the link, I could see a webpage which had a small PHP code written on it, it was evident that this was the source code of the `index.php` file being used here.




```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__);
}

?>
```

It was basically a bunch of if else statements, where it was taking an input from user in the variable `'hash'`. It was clearly visible that we need to fulfill a specific condition to include the `flag.php` file and get the flag.

First, we must understand what a hash value is. A hash value is a fixed-size string or number generated by a hash function from input data of any size. It's used for efficiently indexing

data, ensuring data integrity, and enhancing security. Hash functions, such as MD5, SHA-1 or SHA-256, produce a unique hash value for different inputs. They are crucial for verifying data integrity, protecting passwords, and managing large data sets efficiently. Though collisions can occur (i.e. more than one input can have the same hash) it is very difficult to brute force those collisions, that is why hashes cannot be reverse engineered.

So now let's see what the condition for getting the flag is:

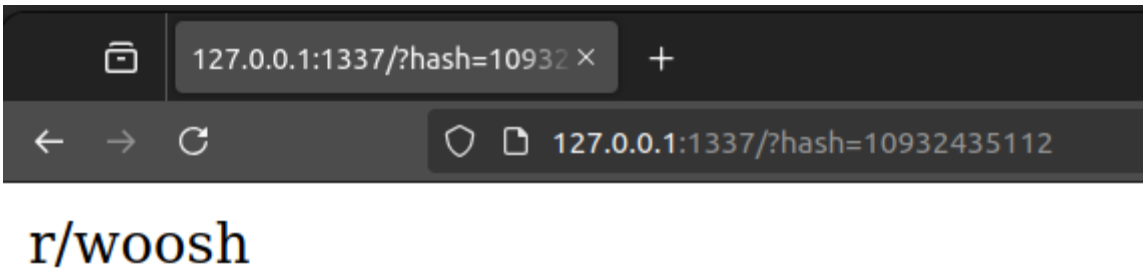
```
$hash = sha1($_GET['hash']);
$target = sha1(10932435112);
if ($hash == $target) {
        include('flag.php');
        print $flag;
} else {
        print "ehax{sad}";
}
```

It is calculating the SHA-1 hash of the input we are giving and stores it in `'hash'`. Similarly, it calculates SHA-1 for `10932435112` and stores it in `'target'`. Then the if condition checks if `'hash'` and `'target'` are equal then it will show the flag, else if will print `ehax{sad}`.

From this it is obvious that we just need to enter the number `10932435112` so that the hash of input matches the target value.

To input value to a variable in PHP we can do that using the URL/Address Bar:

http://sampledomain.com/?variable_name=value



So, when I give the input for `'hash'` as `10932435112`.....Umm, I didn't get the expected output, rather we get something totally different, i.e. it prints `r/woosh`.

Now if we look carefully, there is another conditional statement before the hashing takes place, that is:

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

This means if our input is equal to `10932435112`, the script will stop execution while printing `r/woosh` and the 2nd conditional statement won't even be executed.

So, to get to the `flag.php` we need to enter a value which is not equal to `10932435112` but whose hash is equal to the hash of `10932435112`. Sounds quite ironic right?

Then a thought encountered my mind, although I mentioned above that hashes cannot be reverse engineered and it is very difficult to find collisions, but SHA-1 specifically is considered to be a very weak hashing algorithm, that is because of its small output size collisions have been achieved for this algorithm. So, I thought of finding the input which would collide with the hash of `10932435112`. But it was soon that i realized, although collisions have been achieved but that requires a lot of computational power to brute force all the possible inputs and that is not feasible for us on our daily use laptops.

Then i was just observing the code carefully and i noticed something, when the first conditional statement is being checked we are using the `===` operator, and in the 2nd conditional statement we are using the `==` operator. And this rang a bell in my mind.

In PHP == operator checks if the values on both sides equal, if so then it returns true else false. On the other hand, === operator also checks if the values are equal, but it also checks if the data types are equal or not. When we use the == operator implicitly converts the type of operands and compares them, on the other hand === would just return false if the data type on both the side of the operator are different. This is known as type juggling and is more like a feature of PHP rather than a vulnerability but can be exploited if used the wrong way (like in this case).

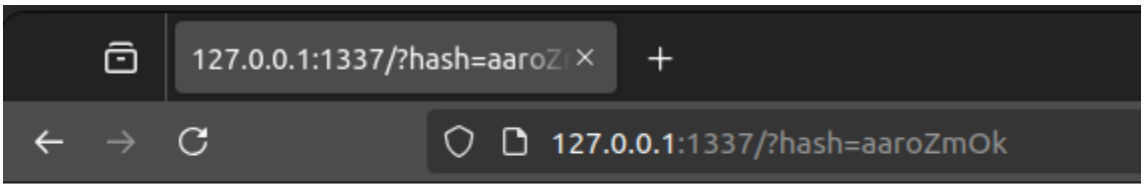
This seemed to be a feasible exploit to bypass the first condition and execute the 2nd one.

SHA-1 of 10932435112 is 0e07766915004133176347055865026311692244

For the 2nd conditional statement == operator is being used which converts, the string "`0e07766915004133176347055865026311692244`" into integer, it would take integers only till the first character is encounter, thus it would be equivalent to comparing with 0.

Thus, we would need such an input which is not equal to `10932435112` but the hash starts with 0. There is a long list of such words on the internet whose hash starts with 0, for instance SHA-1 of **`aaroZmOk`** is **`0e66507019969427134894567494305185566735`**. (https://github.com/spaze/hashes/blob/master/sha1.md)

When we give this as input, this would not be equal to `10932435112` and would thus not execute the `die()` statement and go on to check the 2nd condition. Here its hash would be converted to integer which would be equivalent to 0, which is compared with 'target' whose integer conversion is also 0, and thus the condition would evaluate as true, giving us the flag.



ehax{l3mm3_jUggl3_dem_b4lls}