Hard Login

This challenge relies on a strange "feature" in PHP known as type juggling.

The challenge:

Better put your thinking cap on for this one. It's the hard one, after all.

http://redacted-url:64492/

Follow the link and you'd be taken to another login page, this time with a ugly red background. The source code hides the clues for how to get the password:

```
<!-- TODO - verify login code logic
            dont worry about these comments being public cus
              i hashed the password! duh!
  $username_hash = '21232f297a57a5a743894a0e4a801fc3'; //login: admin
  $password_hash = '0e549193047796524270260661450500'; //password:

  if (md5($username) === $username_hash) {
      if (md5($password) == $password_hash) //fixme
        $msg=$flag;
      else
          $msg="incorrect password!";
  } else
    $msg="bad username!";
-->
```

There is a username and password hash (and the plaintext username so you don't have to worry about cracking that) and a snippet of the source code. It's not likely you'll find the source code like this in comments in the real world, but who knows.

Digging into the code you'll see a //fixme comment, which is for the password hash comparison. IF you're unfamiliar with hashing you may want to google it to get an understanding. The md5 hash of the password is being compared to the pre calculated hash in the code. Md5 is outdated and shouldn't be used, but we're not necessarily looking for a flaw in md5 in this challenge. The fact that it uses md5 does make it much easier to generate a hash that will work, though. The vulnerability we can utilize is the 'loose' comparison operator: '==' (as opposed to the 'strict' operator: '==='. The concept is better described here: https://www.owasp.org/images/6/6b/PHPMagicTricks-TypeJuggling.pdf

This is a challenge that most likely requires the player to do a fair bit of research and is meant to be harder than most of the others. Basically, you don't need to find an exact hash. Any hash that starts with 0e and ends with all digits will match with the given password hash. The PDF should help you understand how that works. Here is the actual password and hash used on the website:

plaintext:"R34llyV4rySUp3rG00dP455sW0rd670773149",
md5:0e549193047796524270260661450500

And here is another bit of plaintext and its hash that follow the same 0e + digits format

plaintext:"240610708", md5:0e462097431906509019562988736854

Even though both of these hashes do not match, when compared with the php '==' operator, they WILL match and '240610708' will work when used as the password.

Here is some Python code that will generate a plaintext/hash combo like this. It takes a few minutes to give the first result on my laptop - your mileage may vary. If you do a bit of googling for php magic numbers you may find something that works without having to crunch it yourself!

```
import hashlib

target = '0e'
n = 0
#word = "R34llyG00dP455sW0rd"
while True:
    plaintext = str(n)
    #plaintext = str(word) + str(n)
    hash = hashlib.md5(plaintext.encode('ascii')).hexdigest()
    #if hash[2:].isdigit():
    if hash[:2] == target and hash[2:].isdigit():
        #print(hash[:2] + '    ' + hash[30:])
        print('plaintext:"' + plaintext + '", md5:' + hash)
        #break
    n = n + 1
```