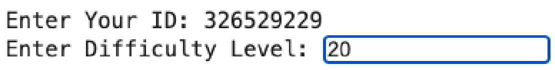
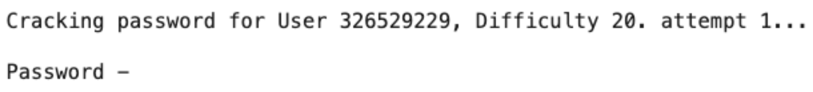
1. At first you are presented with input boxes for your username(id) & desired difficulty level.



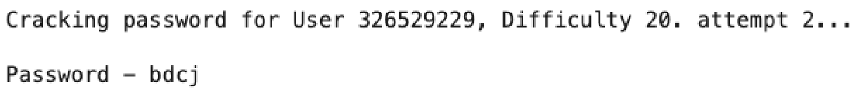
```
Enter Your ID: 326529229
Enter Difficulty Level: 20
```

After that the Attack will start working…

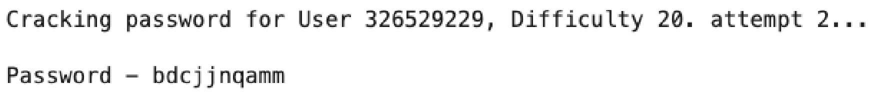


```
Cracking password for User 326529229, Difficulty 20. attempt 1...

Password -
```

Each time the attack decides on a character, it joins the password output:

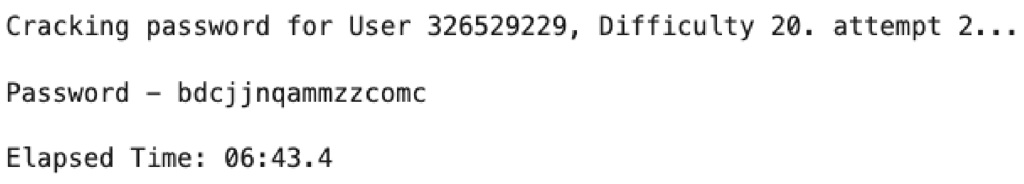


```
Cracking password for User 326529229, Difficulty 20. attempt 2...

Password - bdcj
```



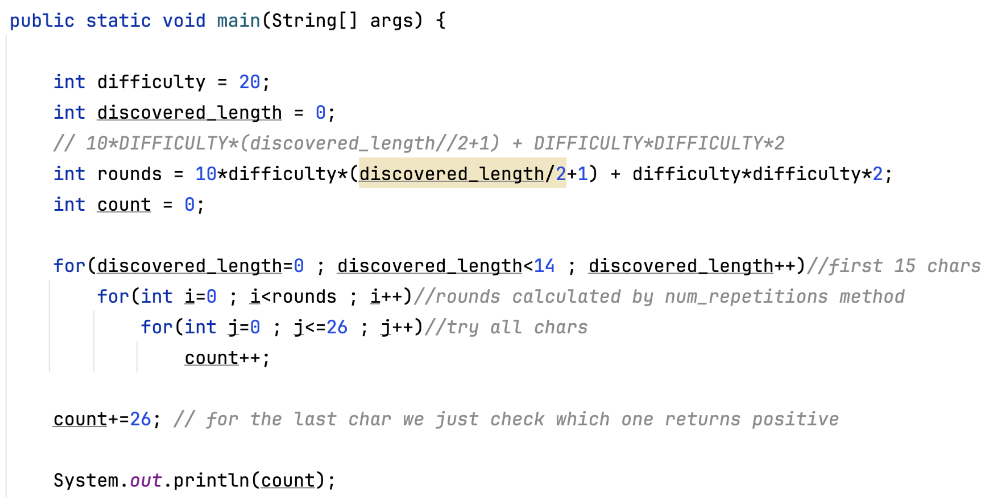
```
Cracking password for User 326529229, Difficulty 20. attempt 2...

Password - bdcjjnqamm
```

Until the password is finally cracked (:



```
Cracking password for User 326529229, Difficulty 20. attempt 2...

Password - bdcjjnqammzzcomc

Elapsed Time: 06:43.4
```

2. For a password of length 16, with a charset sized 26,
Bruteforce will take about 26^16 attempts which is around **639,909,179,494,039,552.**
My Timing attack takes exactly **4562 tries for difficulty 1**, and **378,026 tries for difficulty 20.**
(I used this calculation code which is similar to my attack code)



```java
public static void main(String[] args) {

    int difficulty = 20;
    int discovered_length = 0;
    // 10*DIFFICULTY*(discovered_length//2+1) + DIFFICULTY*DIFFICULTY*2
    int rounds = 10*difficulty*(discovered_length/2+1) + difficulty*difficulty*2;
    int count = 0;

    for(discovered_length=0 ; discovered_length<14 ; discovered_length++)//first 15 chars
        for(int i=0 ; i<rounds ; i++)//rounds calculated by num_repetitions method
            for(int j=0 ; j<=26 ; j++)//try all chars
                count++;

    count+=26; // for the last char we just check which one returns positive

    System.out.println(count);
```

3. To make my program as fast as possible, I first started by using a lot of threads.
   Since most of the thread’s lifetime in my program is waiting on I/O, there is no problem using 1000+ threads even if the computer only has 6 cores.
   But with too much stress on the server I observed errors when trying to communicate with the server. So I tried to minimize the number of tries-per-char & threads by using more accurate measurements- the trimmed-mean statistic allowed me to get great accuracy with less tries than before.
   In addition to that, instead of using Pycurl’s total_time measurement, I used:
   STARTTRANSFER_TIME - the time until the first response byte was received.
   PRETRANSFER_TIME - the time that took pycurl to set up the request.
   These allowed me to get a measurement closer to the server’s processing time.