BCrypt Hashes

Mark Nakamae

Cal Poly SLO

June 2020

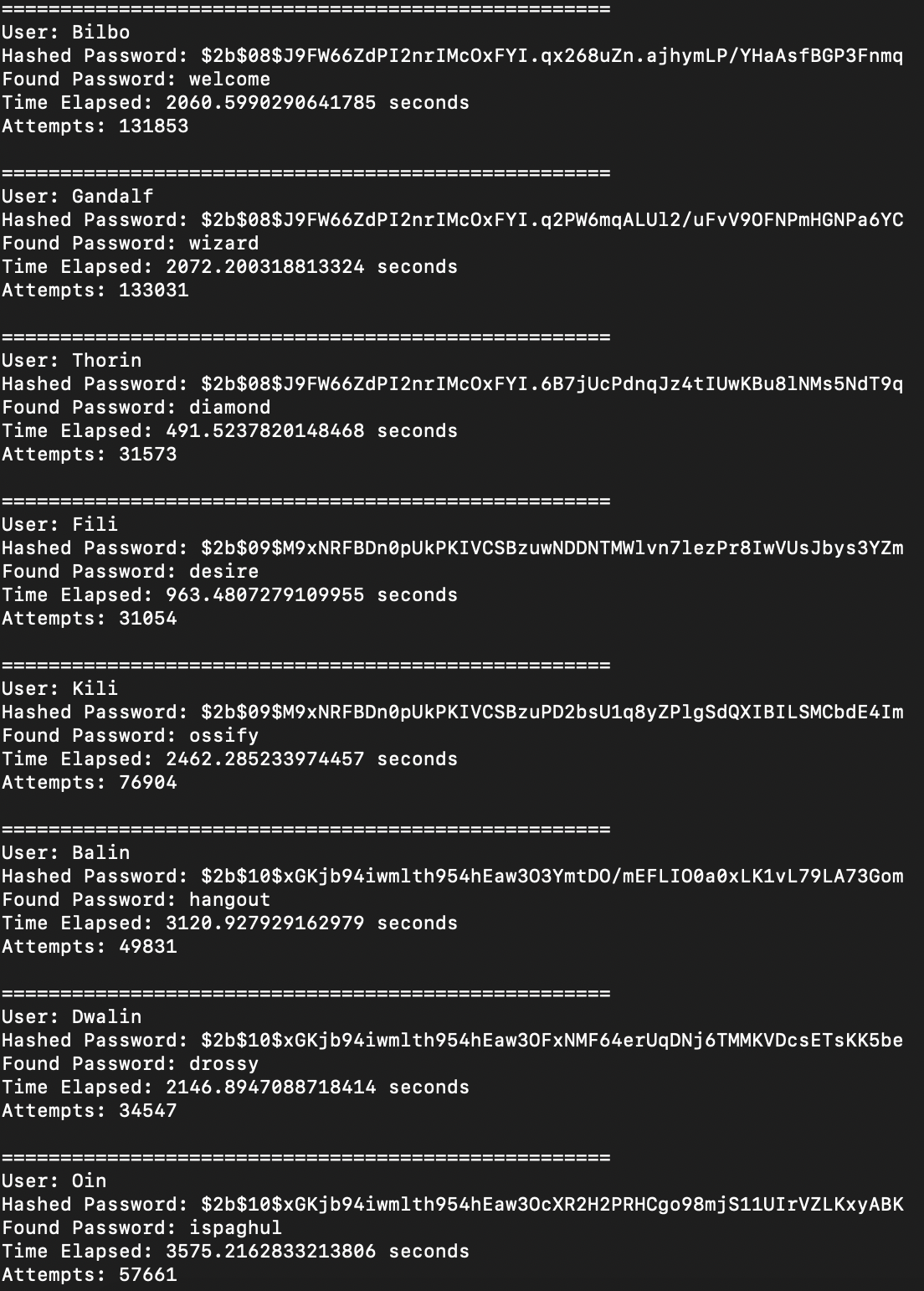
Description

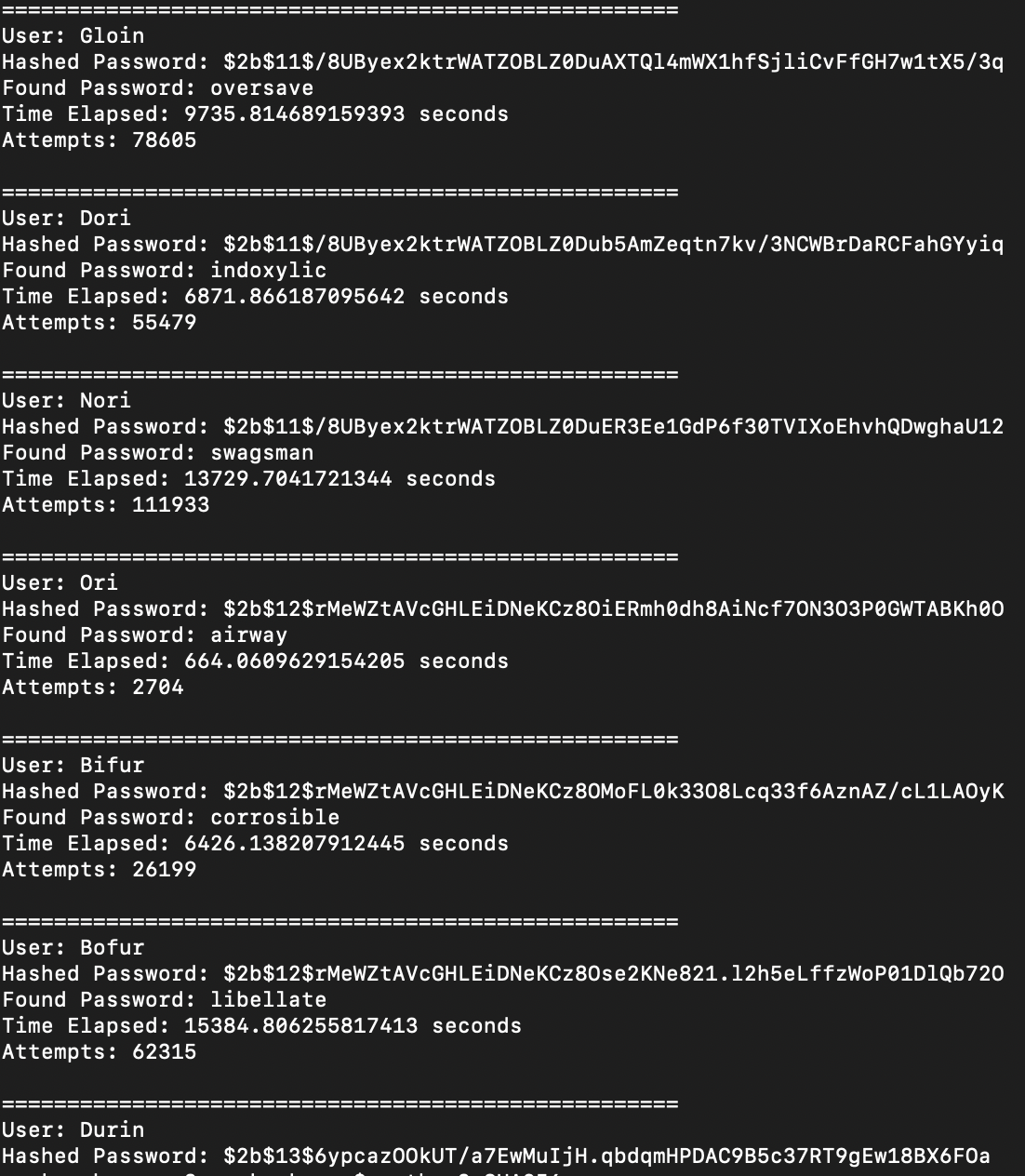
[bcrypt](https://en.wikipedia.org/wiki/Bcrypt#:~:text=bcrypt%20is%20a%20password%2Dhashing,presented%20at%20USENIX%20in%201999.) is a hashing algorithm often used on passwords or other highly confidential information. This algorithm is used because it takes very long to break given the one way nature of the computations. Additionally, when salts are added on to the passwords, they become even more complex to break.

This project takes a given set of user/encrypted password combinations, and decrypts the passwords by brute forcing the correct guess using the PyCrypto library. To reiterate the how difficult these computations are, the overall runtime to break all the passwords was approximately 31.4 hours and took nearly 1.05 million total computations.

Breaking Real Hashes

Breaking bcrypt Results





|  |  |  |  |
| --- | --- | --- | --- |
| **User** | **Password** | **Time Taken (s)** | **Number of Hashes** |
| Bilbo | welcome | 2,060.599 | 131,853 |
| Gandalf | wizard | 2,072.200 | 133,031 |
| Thorin | diamond | 491.524 | 31,573 |
| Fili | desire | 963.481 | 31,054 |
| Kili | ossify | 2,462.285 | 76,904 |
| Balin | hangout | 3,120.928 | 49,831 |
| Dwalin | drossy | 2,146.895 | 34,547 |
| Oin | ispaghul | 3,575.216 | 57,661 |
| Gloin | oversave | 9,735.815 | 78,605 |
| Dori | indoxylic | 6,871.866 | 55,479 |
| Nori | swagsman | 13,729.704 | 111,933 |
| Ori | airway | 664.061 | 2,704 |
| Bifur | corrosible | 6,426.138 | 26,199 |
| Bofur | libellate | 15,384.806 | 62,315 |
| Durin | purrone | 43,275.634 | 160,370 |

*Given your results, how long would it take to brute force a password that uses the format word1:word2 where both words are between 6 and 10 characters?*

The algorithm used to get these runtimes involved checking words lexicographically, so the time taken would depend on word1 and word2’s position in the alphabet. Additionally, this time depends greatly on the work factor of the salt, in which larger work factors generally saw much larger runtimes. That being said, a generalizable equation to find the runtime would be:

(time taken to check word1) \* (time taken to check word2)

This is because to brute force this password, you would need to check every possible combination of word1 and word2, which would take (as worst case scenario):

(number of words) \* (number of words) \* (time to hash one password)

or

**(number of words)2 \* (time to hash one password)**

*What about word1:word2:word3?*

Similarly to the question above, the worst case generalizable equation for this would be:

(number of words) \* (number of words) \* (number of words) \* (time to hash one password)

or

**(number of words)3 \* (time to hash one password)**

*What about word1:word2:number where number is between 1 and 5 digits? Make sure to sufficiently justify your answers.*

This equation now follows the format:

(number of words) \* (number of words) \* (number of numbers) \* (time to hash one password)

There are \_\_\_ different combinations of numbers with 1, 2, 3, 4, or 5 digits. This means the equation is now:

(number of words) \* (number of words) \* (101 + 102 + 103 + 104 + 105) \* (time to hash one password)

or

**111,110 \* (number of words)2 \* (time to hash one password)**

Code

**import** time

**import** bcrypt

**import** base64

**from** nltk.corpus **import** words

#generate a hash from given password and work factor

**def** try\_hash(salt, password, work\_factor):

hashed = bcrypt.hashpw(password.encode("utf-8"), salt.encode("utf-8"))

**return** hashed.decode("utf-8")

#brute force guess the password

**def** guess\_pass(work\_factor, salt, b\_hashed):

start\_time = time.time()

**for** word **in** words.words():

#only words with length 6 <= len <= 10 are valid

**if**(len(word) > 10 **or** len(word) < 6):

**continue**

#generate hashed password

tmp = try\_hash(salt, word, int(work\_factor))

#if password found

**if**(tmp == b\_hashed):

**print**("Found Password: " + word)

run\_time = time.time() - start\_time

**print**("Time Elapsed: " + str(run\_time) + " seconds")

**print**("Attempts: " + str(count))

**break**

**def** main():

f = open("shadow.txt", 'r')

#for each password

**for** line **in** f:

#get fields from hashed password

user = line[:line.find(':')]

algo = line[(line.find('$') + 1):(line.find('$') + 3)]

work\_factor = line[(line.find("$") + 4):(line.find('$') + 6)]

salt = line[(line.find('$')):(line.find('$') + 29)]

hashed = line[(line.find('$') + 29):]

b\_hashed = line[(line.find(':') + 1):-1]

#print results

**print**()

**print**("====================================================")

**print**("User: " + user)

**print**("Hashed Password: " + b\_hashed)

guess\_pass(work\_factor, salt, b\_hashed)

f.close()

**if** \_\_name\_\_ == '\_\_main\_\_':

main()