|  |  | **Cybersecurity Lab, CSE3140** |
| --- | --- | --- |
|  |  | **Spring 2022** |
| **Password Cracking** | | |
| **Section #:1** | | |
| **Team #:9** | | |
| **Names: Ali Al Hamadani, Jared Moore** | | |

Passwords exist to be private —it is usually what you know/have to access an asset (a school system, bank account, website, service, etc.). If this private information is revealed by someone else, that person can have complete access to your data, which simply means losing some possessions or getting into trouble. In this project you will be learning about passwords and their role in the security ecosystem.

Creating an account and choosing the password is a critical task for you and the person who is trying to steal your password. Your goal when selecting a password should be to reduce, as far as possible, the likelihood that someone else can discover what it is. On the other hand, people want passwords to be memorizable and easy to type. Unfortunately, these goals often conflict; and many passwords are not too difficult to guess, especially if we can rapidly test many guesses (using a program). Furthermore, password files are often exposed; exposure of a password file of one site may cause exposure to another site, since users often use the same (or related) passwords in different sites and services.

**The password that is simple for you is simple for the hacker to guess. If I wanted to break into an account belonging to someone I know, I’d try as many terms like these as I could think of.** Furthermore, login services should prevent users from using easily guessable passwords. Check this [video](https://www.youtube.com/watch?app=desktop&v=OOgjdQ5N7g8) to learn more on how to protect your password. Also, you may find these two videos helpful, [video 1](https://www.youtube.com/watch?app=desktop&v=FKXYW8erer0) and [video 2](https://www.youtube.com/watch?v=RtUvMJFP_IE)

Notes:

1. For each question, submitted **scripts** are part of the evaluation process. Please include all commands and scripts in the submitted report. Screenshots are acceptable.
2. Quality of the report will be considered in grading (e.g., format and how clearly you present your work).
3. When manipulating files newlines can get appended and make the string comparison fail (you can check their length). This applies to reading from the dictionary as well
4. We recommend using Tmux to allow your script to work in the background.

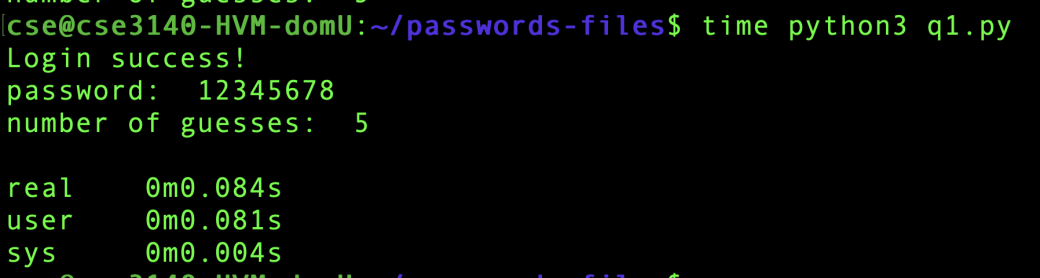
**Question 1 (5 points):**

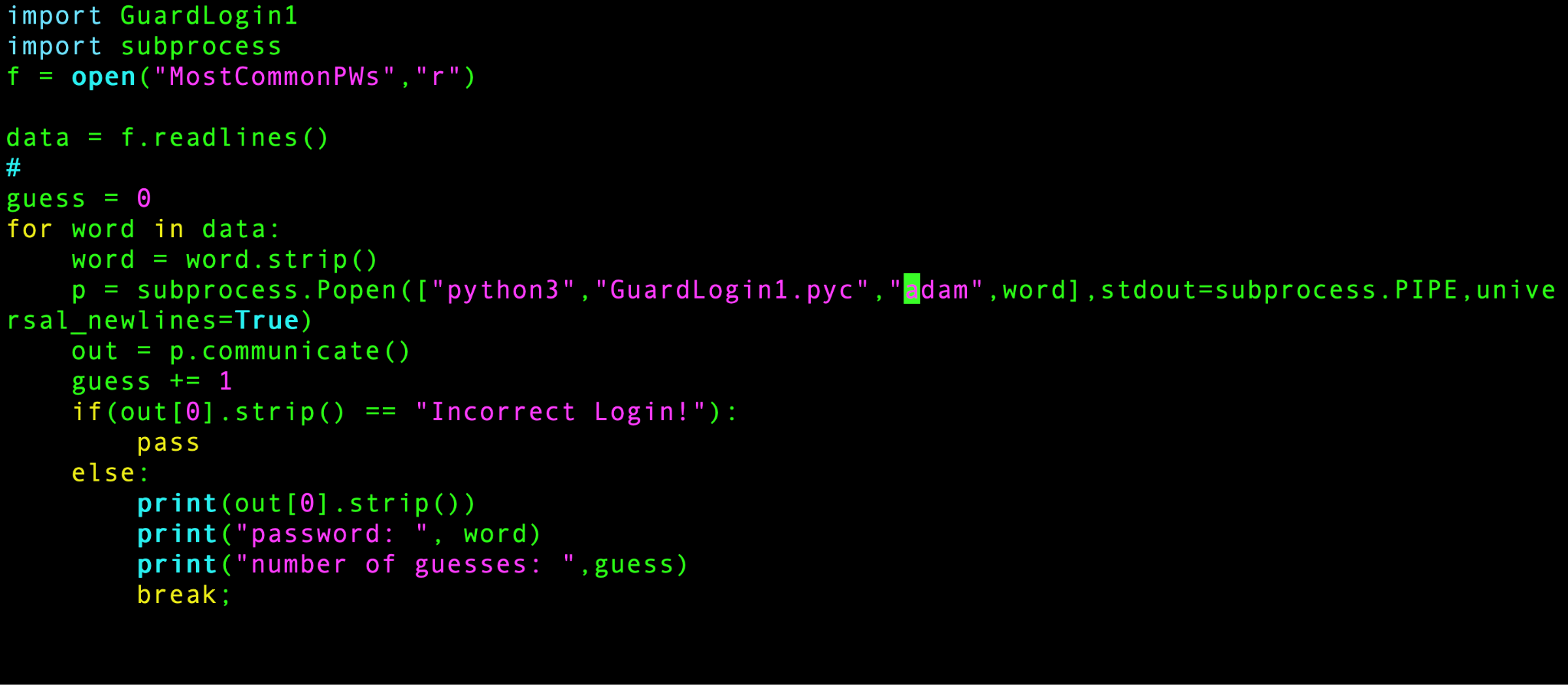
Many users select weak passwords; for example, in 2020, the password ***'123456’*** was used by over 2 million users and exposed over 23 million times, check this [link](https://nordpass.com/most-common-passwords-list/). The file MostCommonPWs lists a few of the most common passwords. To save agent Bond, who was, as usual, captured, you must break into the account of his guard, **Adam L.**, whose userid is simply his first name. Luckily, **Adam** was stupid, and used a quite common password. Use the **GuardLogin1.pyc** utility, to find the password, and provide it in your answer. You will find a **sampleLoginScript.py** to help you understand how **GuardLogin1.pyc** was implemented.

What was the password? **1234567**

How many guesses did you need? **5**

How long did it take to either find the word or run through all of the most common passwords? **.084seconds**

****



**Question 2 (10 points):**

Unfortunately, Bond was captured on his way out. Not as smart as in the movies, is he? Well, let’s save him again. This time, his guard is **Al Capone** ; and the password utility was improved and prevents use of the passwords in the **MostCommonPWs** file. But no problem; the password is surely in the file **Dictionary**, which contains a list of quite-commonly-used passwords. So, you just need to try passwords until you find the right one… Use the **GuardLogin2.pyc** utility as much as you need, to find the password, and provide it in your answer. Hints: (1) **Al**’s username is again his first name, (try small case and capital case) (2) programming is good for you, write some code!

***Note:*** when manipulating files newlines can get appended and make the string comparison fail. Similarly, when reading from a dictionary make sure to remove newlines.

Write a script **(q2.py)**that uses the **dictionary** to guess the password. Answer the following:

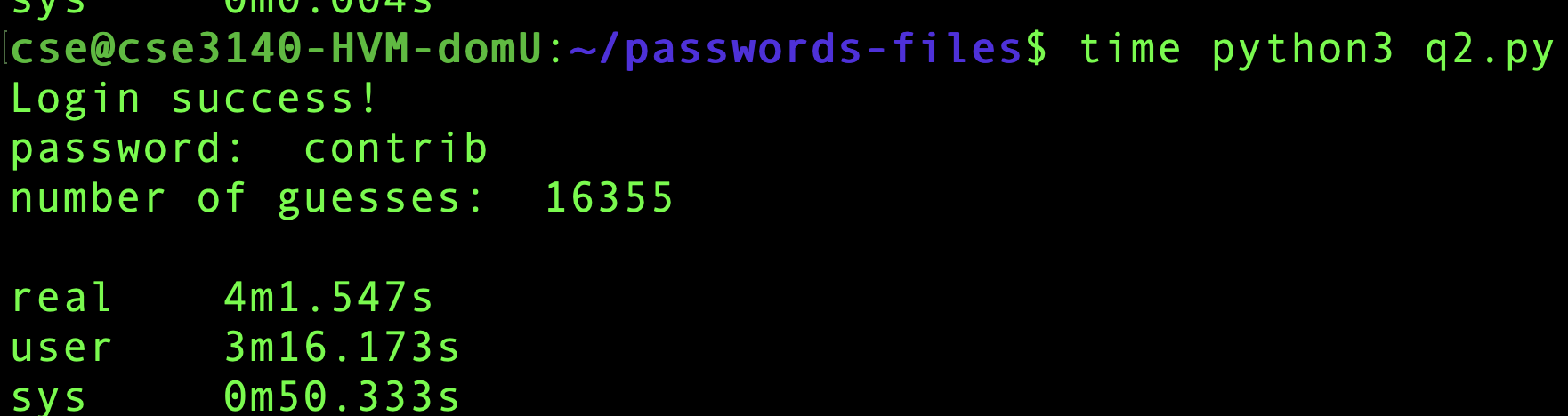
What was the password? **contrib**

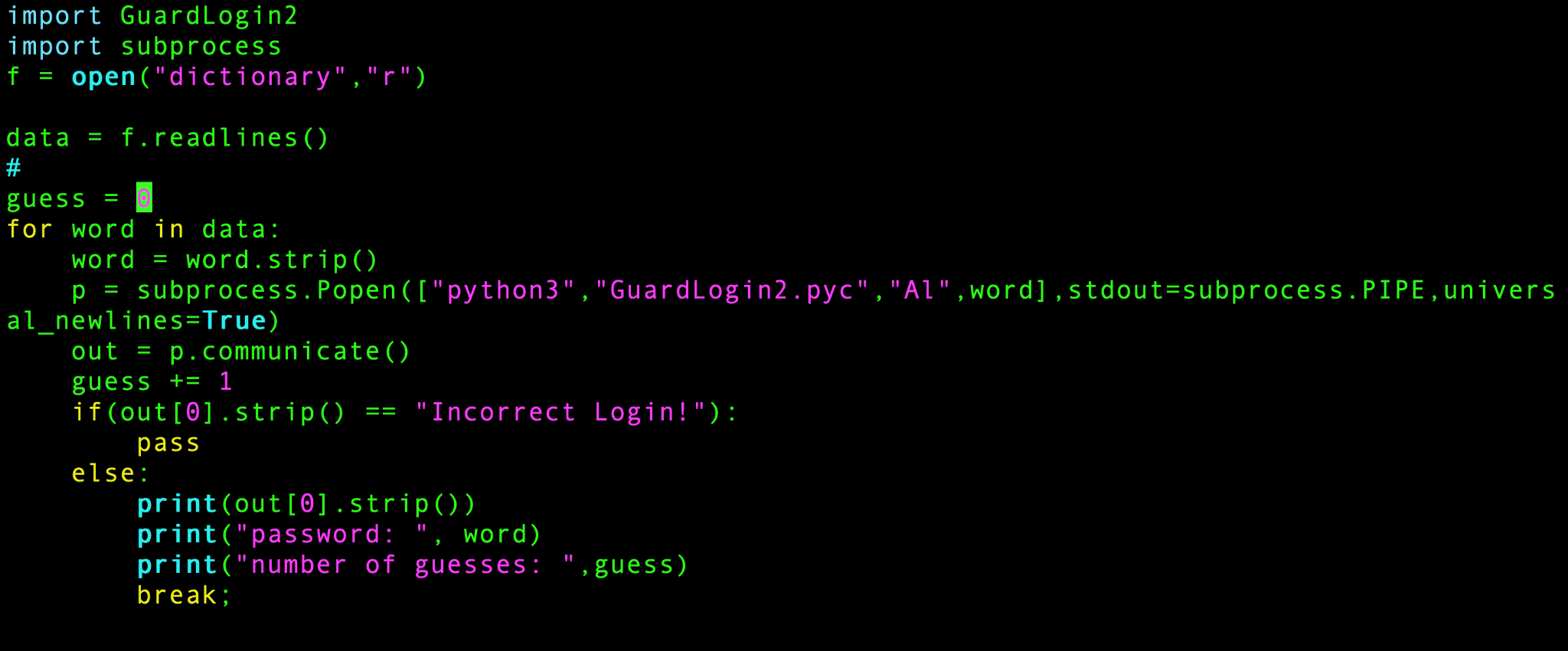
How many guesses did you need? **16355**

The time to execute these guesses. **4 mins, 1.5 seconds**

What is the factor that made this less/more secure from the previous question (#1)?

**The list of potential passwords was larger, and therefore brute-forcing the passwords took more time compared to the smaller passwords file in question 1.**





**Question 3 (10 points):**

Before you manage to get the agent bond out, the guards were able to figure out the risk **Al Capone** is putting on their system. They asked him to reconsider choosing his password. **Al Capone** was fast enough to change his old password with a new one from the same dictionary, but he added in a random position one special character from this list [!,@,#]. Write a script that uses the **dictionary** to guess the password. The new login utility is GuardLogin3. (The script may take some time, feel free to open a new terminal and go to question 4)

Answer the following:

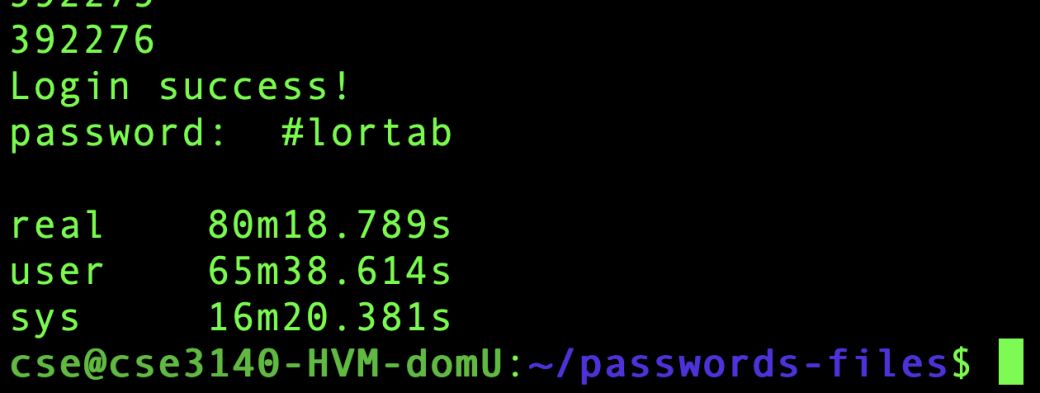
What was the password? **#lortab**

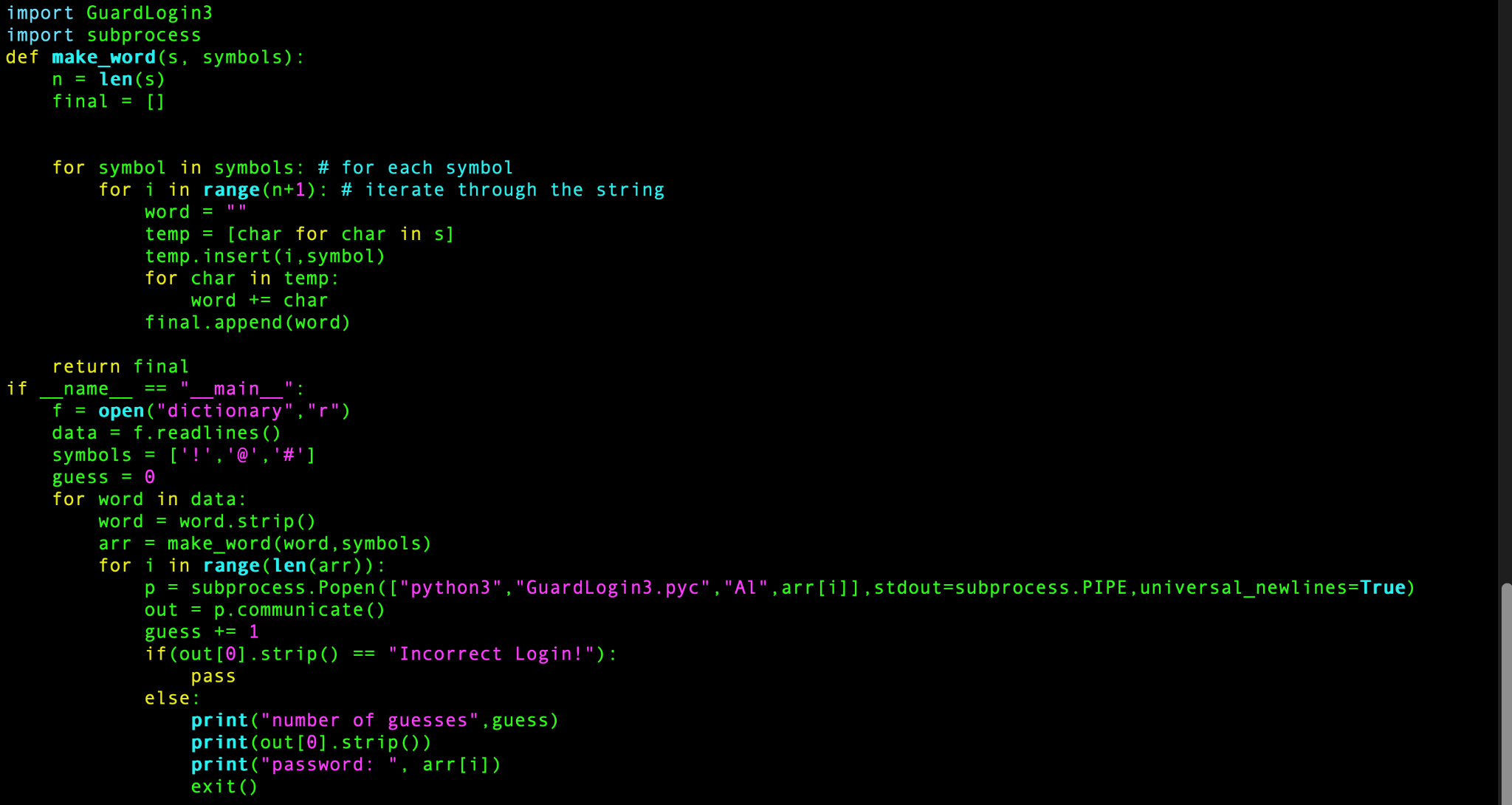
How many guesses did you need? **392276**

The time to execute these guesses. **80m18s**

What is the factor that made this less/more secure from the previous question (#2)?

**The factor that increased security was increasing the number of potential characters needed to be checked against in the dictionary attack. This shows two things, the first is that generally adding special characters increases password security greatly, as it increases the base number of characters (exponentiated by the length) if you were to generate every possible iteration to brute force this password. The second observation we can make is that since this special character was added singularly and he still used a dictionary word password, the number of potential passwords was greatly reduced from that of 30 (26 alphabetical + 4 char) to the power of 7 (len #lortab = 7) to for n (number of words in the dictionary) \* len of each word summed.**



****

**Question 4 (10 points):**

Bad news: Bond was re-captured. Yes, again. And they improved the password utility again: it takes 20 seconds per guess – and we don’t have all day, they are torturing him! The previous solution may not work well enough.

Good news: hackers posted (yet another) password file online; we got a copy of it for you, file name **'LeakedPasswords'**. The password list in the format **(username, password)**

A common, bad practice when choosing passwords is reusing the same password on multiple websites. Let us see if we can find a match that works for any of the guards. Hoping one of the guards reused a password from the leaked passwords files. Guards' names are **Charles Manson**, **Ted Kaczynski**, **Tom Horn**, **Bonnie**, **Clyde**, **Andrew Kehoe**, **Ted Bundy** and **Tim McVeigh**, and whose usernames are just the first names. Surely, at least one of them uses the same userid and password as in the Passwords file, right?

Find it! Quickly!! Provide password and guard-name in your answer. The new login utility is GuardLogin4.

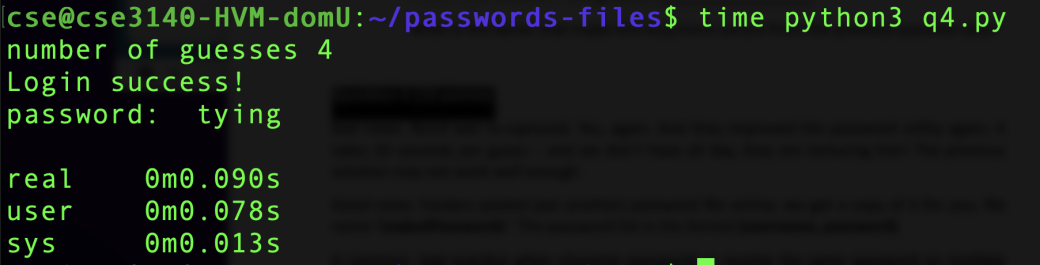
Generate a password file for all students including username and password.

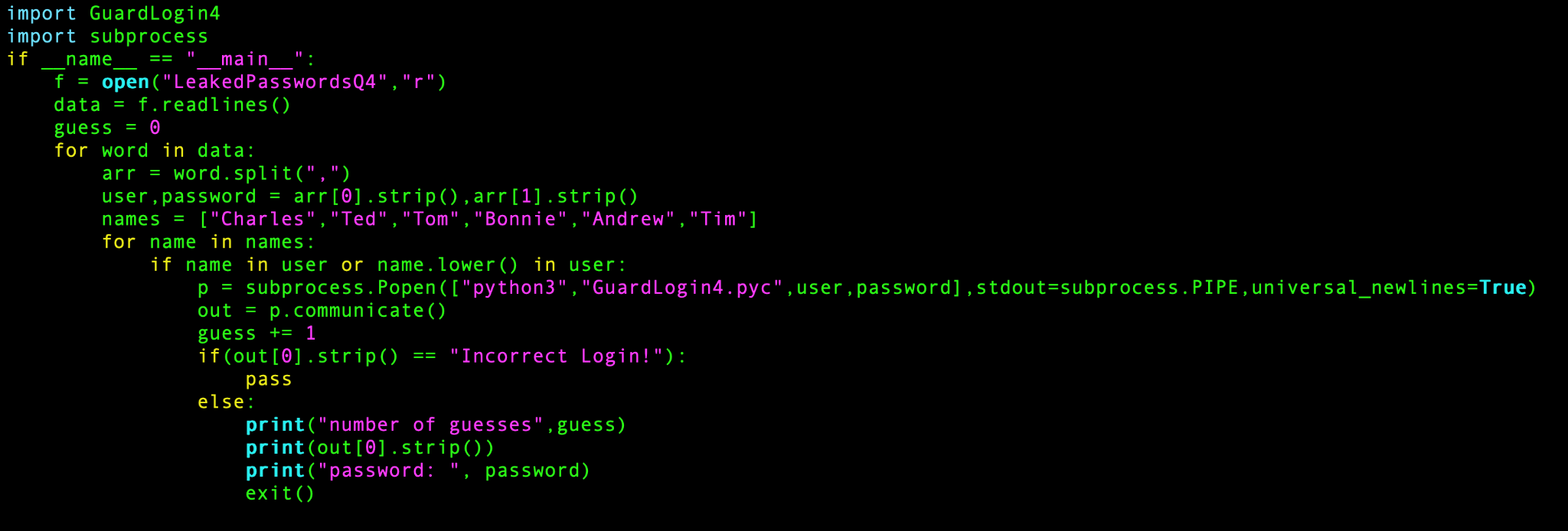
Write a script that uses the **'LeakedPasswords'** to guess the password. Answer the following:

What was the password? **tying**

How many guesses did you need? **4**

The time to execute these guesses. **0m0.090s**

****

****

**Question 5 (15 points):**

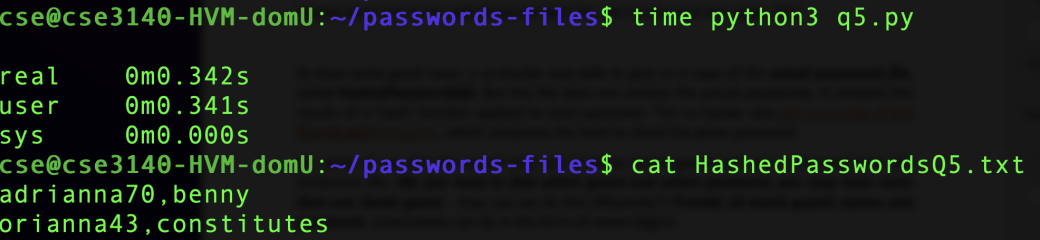
Guess what?? He got re-captured. It’s becoming boring, no? Worse: we tried all the passwords we found in the Passwords file posted by the hacker – none works; obviously, they changed all of them. **Moreover, they replaced all the security guards, you do not know their names now!!!!!!!!!**

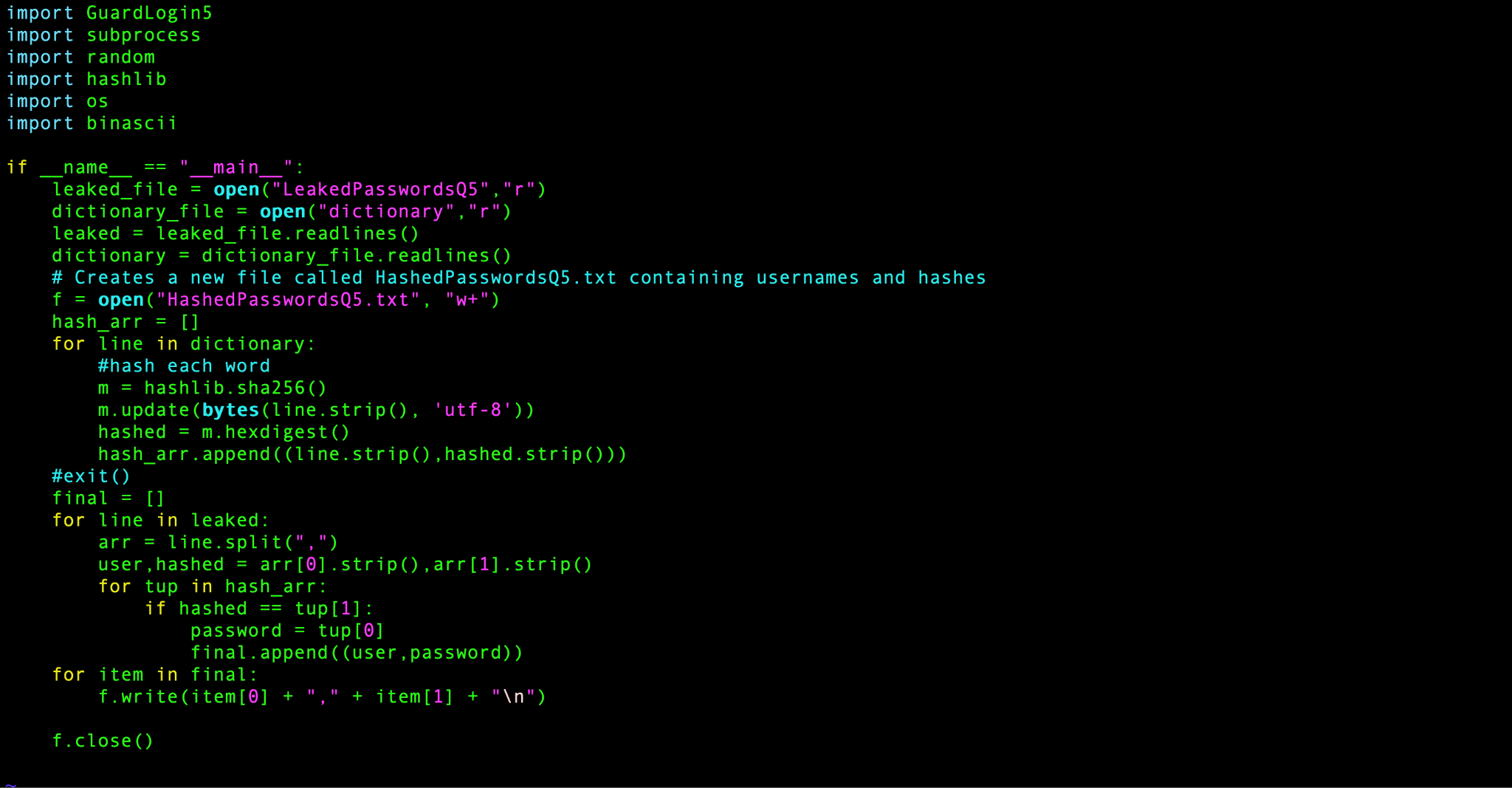
Maybe leave him to his well-deserved fate? No, let’s try to save him again, Ok?

At least some good news: a *co-hacker was able to give us a copy of the* ***actual passwords file***, called **HashedPasswordsQ5**. But this file does not contain the actual passwords, it contains the results of a ‘hash’ function applied to each password. The co-hacker also got us a copy of the **GuardLogin5** program, which computes the hash to check the given password.

Ok, what do we do? We expect that one of these dumb guards will use a password from the Dictionary file. ***We just need to find which guard and which password; you may have more than one dumb guard***… How can we do this efficiently?? **Provide all dumb guards names and passwords**. (Usernames can be in the form of name+digits)

Note: A cryptographic hash is a function that takes arbitrary length inputs to a fixed length string.  In this lab we’ll focus on 256-bit length outputs.

****

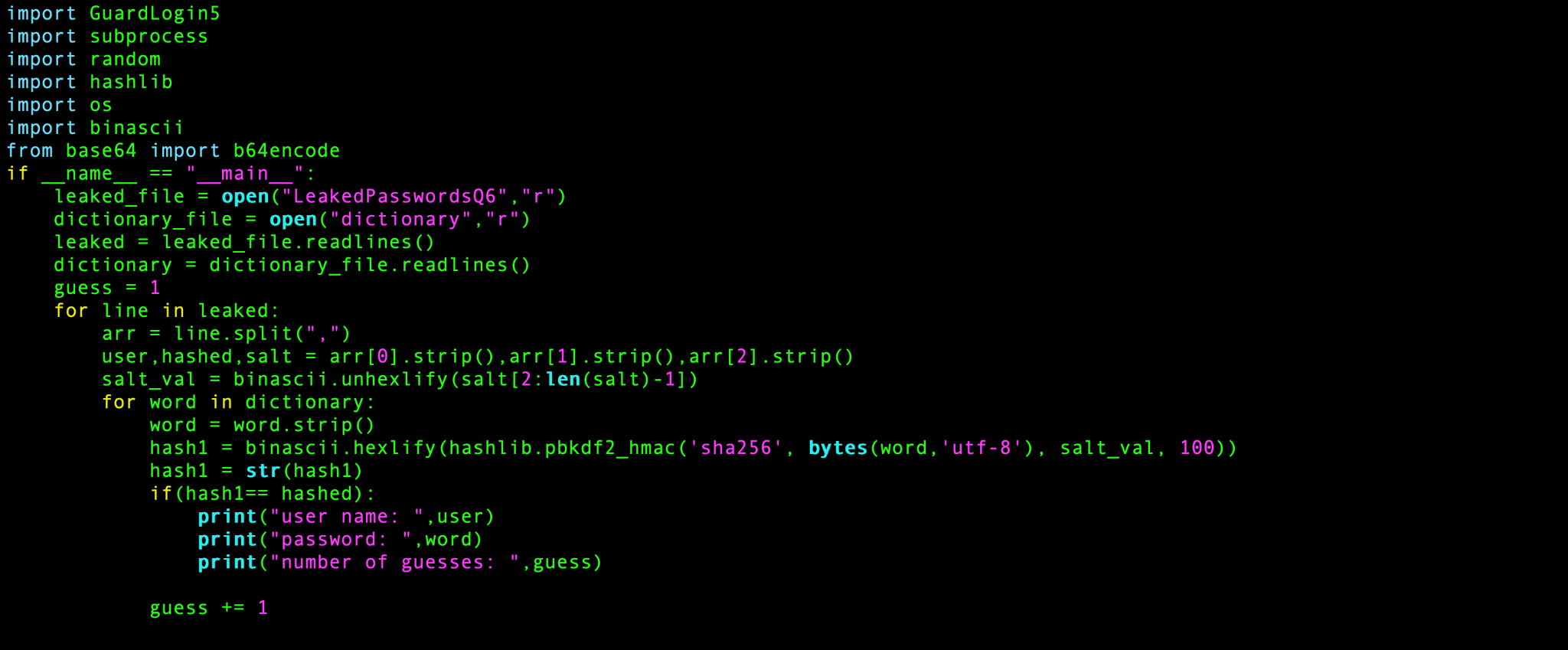
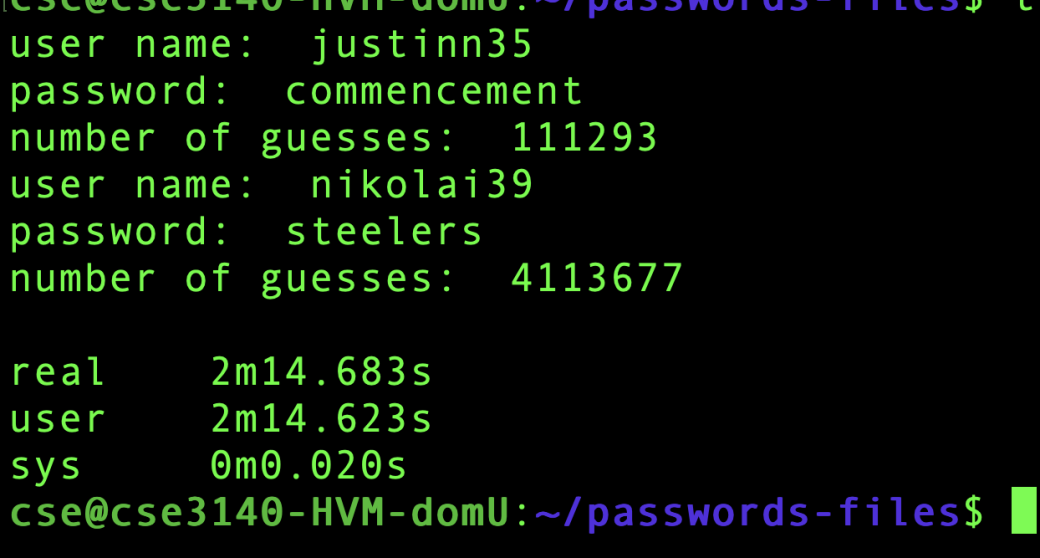


**Question 6 (15 points):**

Oh, not again! He got caught – and the system was further modified! But at least, we received the new password file; it’s now called **SaltedPasswords**. This file contains the hash for each password, almost as before, except that the hash is applied to the **concatenation** of the password and of a random value called `salt’; the `salt’ values are also included in the SaltedPasswords file. And we also have the new **GuardLogin6** program, which computes the hash over the password and the salt…

Can you help? Let’s hope there’s still some dumb guard using password from the dictionary file.

Can you find and provide the password and user-name of that guard?



Can you identify the most important security difference (advantage or disadvantage) in the use of salt (compared to the previous system, hashing without salt)?

**Salt is the flavor of password security. Salting adds extra randomized inputs to the password before hashing, adding extra gibberish inside the password making it harder to decode even when cracked. This technique can be done a few ways, one being adding a set but secret string to a position in all of your users passwords(this is the less secure option). Other options include adding randomized or user specific randomized salts which can make it even more difficult to gain passwords. Despite the use of salts, passwords can still be cracked if a large enough database of passwords and their respected salted hashes have been found, because we can statistically map the frequency of certain matching hashes where individuals have the same password, with a table of common passwords. This can narrow our search for passwords using human behavior and engineering, and is one of the main reasons institutions and sites ask you to use strong and *unique* passwords. If your password shows matches with someone else's, the extra data aside from your own can be used to compromise your password.**

**Question 7 (15 points):** Every time a guard chooses a password agent **Bond** manages to escape. The security guards got crazy about it and decided to use a password generator to select a strong password, namely, **“creation.py”**. Someone from your team was able to get that program “creation.py”, and was able to capture the guard generated hash, stored in another file named ***hash\_to\_crack7***.  **“creation.py”** is a python script that was used to generate the **hash\_to\_crack7** file. Study the file and answer the following questions.

1. How was your string ***hash\_to\_crack7*** created?

**It took a random word from the dictionary and it picked a random spot in the word to insert a symbol, and another random spot in the word to insert a number. Similar to Q3.**

1. How many possible passwords could be created by ***creation.py***?  It may help you to know the average length of a word in dictionary file.  Show your work.

**Each line has a line feed character so the number of line feeds is equal to the number of lines.**

**So: the number of characters(including line feed) - number of lines = Number of characters(excluding line feed) = 155418 - 20000 = 135418**

**Now we can divide this number, by the number of lines, the average length of a word.**

**135418 / 20000 = 6.77 ~ 7.**

**1.The number of choices/spots to insert a number in a word is length(word) + 1.**

**So for our example it would be 7+1 = 8.**

**2. The number of choices for a number is 0-9, so 10 choices.**

**3. The number of choices/spots to insert a symbol is equal to the new length of the word(after inserting a number) + 1 = 8 + 1 = 9.**

**4. The number of choices for a symbol is 10.**

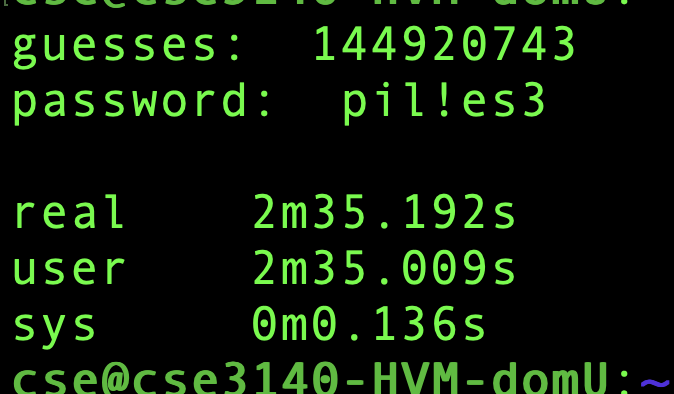
**So multiplying steps 1,2,3,4 we get 8 \* 10 \* 9 \* 10 = 7,200, this is the number of possibilities that can be created from creation.py**

1. Write a program to find what password was used to create ***hash\_to\_crack7***.  You can try to algorithmic reverse SHA256, this may or may not be a good idea.  What are your other options? You may use any or all of the code in ***creation.py*** to help you.

**The other option is making the combination of password from creation.py and hashing it and see if the hashes match.**

   
Include your code and report the following:

1. The password: **pil!es3**
2. How many guesses you needed **144920743**
3. The time to execute these guesses. **2m035s**

****

**Question 8 (15 points)**:

From bond’s exercises, questions 5 and 6, you already know how to deal with a leaked password file for all logins to a system. Use the website [HaveIbeenPWNed.com](https://haveibeenpwned.com/) to see if your email (and password) got exposed in a previous leakage incident, check the link [here](https://www.upguard.com/blog/biggest-data-breaches?utm_campaign=newsletters-2021&utm_medium=email&_hsmi=107357784&_hsenc=p2ANqtz--Hn2he_UmopR9KN7YD-yrg2GXIvc8E6WScdw2zmgoD5UHtOQBbPLH4S-hFzYiggPDJ6BQxkeetO_DnjMsQvXi-0VKxttHU7W7NtMgQSqsTKqf15qg&utm_content=107308563&utm_source=hs_email) and [here](https://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/)  for some examples of a few famous data breaches.

In this exercise we would like you to see how many hashes you can recover from a LinkedIn password breach incident. One famous data breach was the LinkedIn accounts of about 170 million people that were compromised in 2012. The hashed password file unexpectedly appeared on the dark web in 2016. The breach revealed that linkedin was using **unsalted SHA1** for hashing the passwords. The LinkedIn hacker, a Russian, was sentenced to over seven years in prison by a federal jury on September 29th, 2020.

To start, you will find a file named **"LinkedIn\_HalfMillionHashes.txt"** in your project folder, namely, passwords-files; usernames were removed for privacy. The hashes are all in SHA1, just like in the previous questions. The file contains multiple hashes that you will work on cracking. This time you can use a powerful tool that is used widely to crack hashes: **"hashcat"**. You can learn it by typing **"hashcat -h"** to see the manual. You can also watch this [**video**](https://www.youtube.com/watch?v=EfqJCKWtGiU) to understand how to use the tool. You can use the **dictionary** file you used before to crack these hashes. A better wordlist, named “rockyou.txt,” is also available to use.

Try to find passwords that have one-to-one matches in the “**dictionary”** fileand **“rockyou.txt”** file.

1. Use the “**dictionary”** fileas your source for the wordlist to try cracking **LinkedIn\_HalfMillionHashes.txt**
   * Write the command you used.

hashcat -m 100 -a 0 ./LinkedIn\_HalfMillionHashes.txt ./dictionary

* + How many hash passwords were you able to recover? List the 3 top passwords on your list.

5683 recovered. Top 3 were search, information, contact.

* + How long did it take?

9 seconds.

1. repeat the previous step using the **“rockyou.txt”** 
   * Write the command you used.

hashcat -m 100 -a 0 ./LinkedIn\_HalfMillionHashes.txt ./rockyou.txt

* + How many hash passwords were you able to recover? List the 3 top passwords on your list.

144724 recovered. Top 3 were 123456, 123456789, iloveyou

* + How long did it take?

29 seconds

Let’s try to be more innovative and make use of how users usually pick their passwords. Try to make use of the **rule sets and masks features** and rerun your hashcat. Use only the **“rockyou.txt.”** For each command you try list:

hashcat -m 100 -a 0 -r /usr/share/hashcat/rules/best64.rule LinkedIn\_HalfMillionHashes.txt rockyou.txt

Recovered 205,578

1m53s

hashcat -m 100 -a 3 -1 ?u -2 -?l?u?d -3 ?d ./LinkedIn\_HalfMillionHashes.txt ./rockyou.txt

Recovered 205800

* How many extra Hashes you were able to recover
* How long did it take to recover these hashes

*What do you think LinkedIn could have done better than just using unsalted SHA1, to manage these passwords?*

***Using a better method of encryption, such as SHA256, salting the passwords at the bare minimum to prevent easy and fast dictionary attacks, and ensuring users used passwords which were not dictionary words or were unique enough password policies with letters, symbols, and numbers, to have a fighting chance. Either one of these things, or all things combined, could have changed the outcome for linkedin’s cybersecurity.***

**Valuable Notes:**

* hashcat will report how many passwords it “recovered” when it finishes.
* Keep a backup of the original file of hashes (we saved one for you in the **Documents** folder)
* You can use the [--remove flag](https://haveibeenpwned.com/Passwords) option to remove cracked hashes from the input file. As a result, if you run the commands more than once without changing anything, it won't crack anything after the first time.
* You may need to remove ~/.hashcat/hashcat.potfile to redo the crack
* You can use the wc -l or tail -f to monitor the changes in the cracked file and the output file.

**Question 10 (5 points)**: Why does any system, that requires login, always wants your password to be “difficult for others to guess”? What could happen if it is not? State at least 4 rules to consider (Do and do not while selecting a password) that makes a password hard to guess.

Some issues to consider while writing: 1) who is trying to guess your password? 2) What do they know? 3) How long do you think it takes a computer to try a guess?

**Systems want a password to be “difficult for others to guess” for a few reasons. The first of which is being difficult to guess means being difficult to brute force. If a password is extremely difficult to guess, by either using a wide range of characters, having a long length, and not using dictionary words or combinations of dictionary words, it makes the narrowest table which could contain some combination of values extremely large, as the length, the character count alone will set the upper limit for what a computer would be able to calculate as the set of passwords to try. Using common dictionary words put together, not using alternative characters, or using common passwords for others you know or between accounts poses a risk of deanonymizing and making your password easier to crack. If your password shows up naturally in a small premade list of passwords, you know that it's potentially dangerous. A second potential vector of attack is not directly from brute force, but instead social engineering. In this situation, a password may be conventionally strong, however with other pieces of information about a person's life such as the names of their pets, children, wife, and potentially important dates in their lives (birthdays, weddings, etc) or patterns. If you are not simply being targeted on mass scale, but instead being specifically targeted, a pointed attack could attempt to find vulnerabilities by adding these specific bits of information to make a hashcat mask and text file that can be more tailored to an individual and potentially find security loopholes. For these reasons, I will give 4 rules to consider about making passwords:  
1. Increase the length and range or characters for your passwords, this will make the passwords much more difficult to brute force and take longer to generate a dictionary for.**

**2. Do not use Common Passwords. A general rule is if it's something that could be in a table (dictionary, list of common passwords, table of leaked passwords) or something that could be formed by a combination of multiple tables and/or characters, it's unfit to be used as a password.**

**3. Make your passwords (and security questions for that matter) not specific to you. Using things like birthdays, pets names, partners names, or important live event dates may put you at unnecessary risk. That narrows the pool of potential passwords and makes you more vulnerable if someones targeting you specifically.**

**4. Do not reuse passwords, if you reuse even strong passwords, there's a potential for that password to get breached from a site that improperly stores your passwords with little to no hashing and salting. If that shows up in a public table, it can be used to compromise your other accounts with the same password.**