HW1

Submit Assignment

Due Feb 13 by 11am **Points** 125 **Submitting** a file upload
Available after Jan 30 at 12:15pm

Click here to download HW1 files: HW1.zip

Prerequisites:

You will need to set up and use a specified python environment for this homework. Please refer to this instruction.

Part A: Password Cracking

A colleague has built a password hashing mechanism. It applies SHA-256 to a string of the form "`username,password,salt`", where salt is a randomly chosen value. For example, the stored value for username `user`, password `12345` and salt `999999` is `c50603be4fedef7a260ef9181a605c27d44fe0f37b3a8c7e8dbe63b9515b8e96`. The Python code to generate this is:

```
import hashlib
print(hashlib.sha256("user,12345,999999".encode()).hexdigest())
```

The same process was used to generate the challenge hash `61ef437ca1493baf5ce815a8ca13ec1fba31645f7d85edebac7c60e0aa98b5c6` for user `bucky` and salt `20200128`.

1. Recover the password used to generate the challenge hash above. Hint: The password is an ASCII string consisting only of numeric digits.
2. Give a pseudocode description of your algorithm and the worst-case running time for it.
3. Discuss the merits of your colleague's proposal and suggest how your attack might be made intractable (harder).
4. Put your solutions in the file `solutions.txt`.

Part B: Encryption

Another colleague decided to build a symmetric encryption scheme. These are implemented in `badencrypt.py` and `baddecrypt.py` (see attached `.zip` file) and are designed to encrypt a sample message to demonstrate the encryption scheme. To use these demo programs, run:

```
CT=$(python3 badencrypt.py testkeyfile)
echo $CT
```

```
python3 baddecrypt.py testkeyfile $CT
```

Your job is to assess the security of this encryption scheme. Your solution will be a Python program `attack.py` that takes as input a ciphertext and modifies the ciphertext so that the decrypted message has a different (and more lucrative to the recipient) `AMOUNT` field and still passes the verification in `baddecrypt.py`. The file `attack.py` must do this without access to the key file or knowledge of the key. You can assume the ciphertext contains the sample message hardcoded in `badencrypt.py`.

We will test your solution with original versions of `badencrypt.py` and `baddecrypt.py` and with different encryption keys than the test key provided. To ensure that `attack.py` produces the correct formatted output, you can run from the command line:

```
CT=$(python3 badencrypt.py testkeyfile)
MODCT=$(python3 attack.py $CT)
python3 baddecrypt.py testkeyfile $MODCT
```

1. Complete the attack program `attack.py` (feel free to make modifications to the pre-filled content. The skeleton is provided just to help you out)
2. In `solutions.txt`, describe what is wrong with your colleague's scheme and how it should be fixed so that it will be more secure.

(Your attack script will not have direct access to the key file and should not attempt to gain access to the process memory of `baddecrypt` or any other files to steal the key directly.)

Extra credit: More password cracking

Yet another colleague, to make the password cracking hard, uses hash iteration: SHA256 is iterated 256 times. Something like the following code:

```
h = m.encode()
for i in range(256):
    h = hashlib.sha256(h).digest()
print(h.hex())
```

For example, the input `ironman,password,84829348943` processed with SHA256 iterated 256 times produces the hash `5483d76bc214a60e35a8a068a28912c168ea5aea8d1441559e3568135185d636`. While using the same technique, for the username `bucky` with salt `8934029034`, the challenge hash is `1b2ebfab6e70dcb13f3ff4750d065bab8474dac4dc611df339446071ae3e7977`.

The password is representative of real-world passwords: something complex enough that the person that selected this password would consider using it for a website login, but easy enough to be memorable.

Find the password used to produce the challenge hash. Give a pseudocode description of your algorithm and the correct password in `solutions.txt`.

Hints

- The website has a password policy that requires that the password must have at least 6 characters and atleast three of the four character classes: uppercase letters (`A-Z`), lower case letters (`a-z`), symbols (`` ~`!@#$%^&*()+=_-{}[]\|:;”’?/<>,. ``), and digits (`0-9`).
- You can look at **[CrackStation's password cracking dictionaries (https://crackstation.net/crackstation-wordlist-password-cracking-dictionary.htm)](https://crackstation.net/crackstation-wordlist-password-cracking-dictionary.htm)** for some help.
- It is wise to estimate the running time of your solution before starting it.

Deliverables

- Put all the files (`attack.py, solutions.txt`) in a directory named your wisc ID and compress it.

Grading

- Parts A and B are worth up to 50 points for a total of 100 points for this assignment. The extra credit below is worth up to 25 additional points.

Collaboration Policy

This assignment is to be done individually. You are encouraged to use the internet or talk to classmates for information about tools and setup. Please help your fellow classmates with setup and understanding Python, but don't discuss solution specifics with anyone. Remember, searching for homework solutions online is **academic misconduct**. If two students' submissions are very similar --- for some definition of similarity --- both students will get zero points for this assignment.

HW1 Rubric

Criteria	Ratings	Pts
Correct Password in Part A		20.0 pts
Pseudo Code for Part A		10.0 pts
Running time		10.0 pts
Discussion and improvement		10.0 pts
Part B: code for the attack.py		30.0 pts
Problem with the scheme		10.0 pts
Fix that scheme		10.0 pts
Bonus part: correct password		20.0 pts
Bonus part: correct pseudocode		5.0 pts
		Total Points: 125.0