HW1

Due Sep 24, 2020 by 11:59pm **Points** 100 **Submitting** a file upload
File Types py and txt **Available** Sep 9, 2020 at 11:59pm - Dec 9, 2020 at 11:59pm 3 months

This assignment was locked Dec 9, 2020 at 11:59pm.

Click here to download HW1 files: **HW1.zip**
(https://canvas.wisc.edu/courses/219039/files/14416364/download?download_frd=1)

Prerequisites

You will need to set up and use a specified python environment (python 3.8) for this homework. Please refer to the instructions below:

There are subtle differences between versions of Python, and between versions of Python packages. We need to make sure we're all using the same versions. Otherwise, your code may run differently while the TAs are grading it.

In the following instructions, we assume you have access to a computer with the following programs installed on it.

- python3
- pip
- virtualenv

Note: the CS lab Linux computers satisfy this requirement.

We're going to do the following:

- create a virtual environment for running your cs642 hw1 code
- install specific version of crypto library for that virtual environment.

Open a bash terminal and run the following commands:

```
# Make a directory where you can keep your virtual environments (if you don’t already have one)
$ mkdir ~/envs
```

```
# Make a virtual environment called ’cs642hw1’
$ virtualenv --python=python3.8 ~/envs/cs642hw1
```

```
# activate the virtual environment
$ source ~/envs/cs642hw1/bin/activate
```

```
# install the allowed packages for your environment (You will need the “requirements.txt” file locat
ed in HW1.zip)
(cs642hw1) $ pip install -r requirements.txt
```

You now have a virtual environment identical to the one used by the TAs. You’ll know the virtual environment is active because its name will appear in parentheses to the left of the bash prompt, so run code files by:

```
(cs642hw1) $ python your_code.py
```

When you don’t need the virtual environment, just ‘deactivate’ it

```
(cs642hw1) $ deactivate
```

Packages in the virtual environment should be sufficient for HW1. If you want to use a Python package that isn’t included in the official virtual environment, contact the TAs for approval.

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 following challenge hashes:

a) `53b8da235e6ab04edfe2d73dfd976d4ab26e2bf4e356840ca8104c24a22af139` for user `suleman` and salt `20202293`.

b) `c9808f6d88ffb8089d44b903aed1e09be2d7432be46db8c06c273ca65a0e6fe7` for user `mazharul` and salt `20193833`.

Tasks:

1. Recover the password for both challenge hashes above. *Hint:* Both the passwords are an ASCII string consisting only of numeric digits up to 8 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. Suggest how your attack might be made intractable.
4. Put your solutions under the correct section in the file `solutions.txt`.
5. Also, upload the `pwcrack.py` containing the code to crack the hashes, with clear instructions about how to run it.

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 a slow hash function named `scrypt`. Scrypt is a *password-based key derivation function* that is designed to be computationally intensive (slow). This is because legitimate users only need to perform the function once per operation (e.g., during authentication), and so the computational overhead and the time required is not noticeable. However, a brute-force attacker would likely need to perform the operation billions of times, at which point the time computational requirements become significant and, ideally, prohibitive.
For example, the input `batman,password`, and salt `84829348943` processed with `scrypt` produces the following hash

```
594b32011f597e921b07be213b469a94492ddcdeea84ffea27e2e0392e77f6c59690f1f85b22b8fcb9f551f6613880ef1dc1cc855d600165
b8a285c9a342ad8f
```

```
In [1]: import hashlib

In [2]: message = "batman,password"
        salt = "84829348943"

In [3]: """
        @password and salt must be bytes-like
        @n is the CPU/Memory cost factor
        @p parallelization factor
        """

        h = hashlib.scrypt(password=message.encode(), salt = salt.encode(), n = 16, r = 32, p = 1)

In [4]: print(h.hex())

        594b32011f597e921b07be213b469a94492ddcdeea84ffea27e2e0392e77f6c59690f1f85b22b8fcb9f551f6613880ef1dc1cc855d600165b8
        a285c9a342ad8f
```

While using the same technique, for the username `bucky` with salt `8934029034` (and also keeping `n = 16, r = 32, p = 1)` the challenge hash is

```
0b5445eb04edda3dbe3ce55368908703aec9056efcebb490c75f92afdc4e326c39d4e9fa39c86a2d35252235ffbd86d0f66808abc8ef5678
ef32bede12e7d050
```

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 for Extra Credit:

- The website has a password policy that requires that the password must have at least 6 characters and at least 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)** for some help.
- It is wise to estimate the running time of your solution before starting it.

Deliverables

- Submit all three files `attack.py, pwcrack.py, solutions.txt`.

Grading

- Parts A and B are worth up to 50 points each for a total of 100 points for this assignment. The extra credit below is worth up to 20 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. However, do NOT 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 Passwords in Part A	**20 pts** **Full Marks**	**0 pts** **No Marks**		20 pts
Pseudo Code for Part A	**10 pts** **Full Marks**	**8 pts** **Partial-Marks** Not all password possibilities covered	**0 pts** **No Marks**	10 pts
Running time	**10 pts** **Full Marks**	**0 pts** **No Marks**		10 pts
Improvement	**10 pts** **Full Marks**	**0 pts** **No Marks**		10 pts
Part B: code for the attack.py	**30 pts** **Full Marks**	**0 pts** **No Marks**		30 pts
Problem with the scheme	**10 pts** **Full Marks**	**0 pts** **No Marks**		10 pts
Fix that scheme	**10 pts** **Full Marks**	**0 pts** **No Marks**		10 pts
Bonus part: correct password	**15 pts** **Full Marks**	**0 pts** **No Marks**		15 pts
Bonus Q: Correct pseudocode	**5 pts** **Full Marks**	**0 pts** **No Marks**		5 pts
				Total Points: 120