CNT 5410 - Fall 2023 - Assignment 1: Passwords

Name: Your Name Here

September 11, 2023

This is an individual assignment. Academic integrity violations (i.e., cheating, plagiarism) will be reported to SCCR! The official CISE policy recommended for such offenses is a course grade of E. Additional sanctions may be imposed by SCCR such as marks on your permanent educational transcripts, dismissal or expulsion.

Reminder of the Honor Pledge: On all work submitted for credit by Students at the University of Florida, the following pledge is either required or implied: *"On my honor, I have neither given nor received unauthorized aid in doing this assignment."*

Instructions

Please read the instructions and questions carefully. Write your answers directly in the space provided. Compile the tex document and hand in the resulting PDF as your report.

In this assignment, you will write a few lines Python code. You are encouraged to use Python3. You will need the PyCrypto library to run the provided code. Please refer to resources specific to your operating system and environment for installation instructions[1]. For example, you can use: `pip3 install pycrypto` to install PyCrypto with Python3 under Linux.

Assignment Files

The assignment archive contains the following Python source files:

- `utils.py`. This file contains utility functions needed for the assignment..
- `crypto.py`. This file defines cryptographic functions.
- `rainbow.py`. This file contains a partial implementation of rainbow tables. You will complete this implementation for Problem 2.
- `attack.py`. This file contains attack code used in the assignment.

In addition, the assignment archive contains a `data` directory which includes a dictionary file (`words.list`) and several database dump files in JSON format (`*-dbdump.json`).

Note: You are encouraged to take a look at the provided files. This may help you successfully complete the assignment.

[1] `https://pypi.org/project/pycrypto/` See also: `https://www.pycryptodome.org/en/latest/index.html`.

Problem 0: Bruteforce attack (10 pts)

In this problem, you will mount a bruteforce attack against a stolen database dump of password hashes (`data/simple-dbdump.json`). The code for this problem is already written. (But you are encouraged to take a look to see how it works.)

The bruteforce attack will try all possible passwords of 4 (or less) alphanumeric characters — lowercase only. To perform the attack run the following command[2]:

```
python attack.py problem0
```

1. (3 pts) Which passwords are recovered (for which users)? How long did the attack take (in seconds)? What was the speed of the attack (in hashes/second)?

   Your answer here.

2. (3 pts) What is the total number of *possible* passwords of l *or less* alphanumeric characters — lowercase only?

   Your answer here.

3. (4 pts) Given your previous answers. How long do you estimate it would take to perform the same attack on all passwords of 8 or less alphanumeric characters — lowercase only? (Justify your answer.) Can you confirm your calculation experimentally?

   Your answer here.

[2]You may have to specify to use Python3 `python3` if you also have Python2 versions installed on your system.

Problem 1: Dictionary attack (15 pts)

In this problem, you will perform a dictionary attack against the same list of stolen password hashes (`data/simple-dbdump.json`). You will implement the (generator) function `candidate_dict_generator`() which is (partially) defined in `crypto.py`. The rest of the code is provided.

Your dictionary attack will produce password guesses of the form w **or** $w||s$ where w is a dictionary word, s is a suffix, and $||$ denotes concatenation. Refer to comments in `crypto.py` for more detailed instructions.

Run the following command to perform the attack:

```
python attack.py problem1
```

1. (10 pts) Implement the dictionary attack: `candidate_dict_generator`()!
2. (5 pts) What kind of passwords are recovered and for which users compared to Problem 0? How long would it take to recover the same passwords using the bruteforce attack (Problem 0)? (Justify your answer.)

   Your answer here.

Problem 2: Building a Rainbow table (20 pts)

In this problem, you will perform a Rainbow table attack against the same list of stolen password hashes (`data/simple-dbdump.json`). You will implement (part) of the password lookup rainbow table operation `lookup_rainbow`() in `rainbow.py`. The code to build the rainbow table (`build_rainbow`()) is provided. It will be useful to carefully read the code of `build_rainbow`() and `test_rainbow_attack`() (in `attack.py`).

Use the following to perform the attack:

```
python attack.py problem2
```

1. (15 pts) Complete the implementation of the rainbow table attack: `lookup_rainbow`()! Test your implementation thoroughly. Your attack should recover the same kind of passwords as for Problem 0.
2. (5 pts) What is a false alarm? Explain why it occurs.

   Your answer here.

Problem 3: Understanding Rainbow tables (25 pts)

In this problem, you will run the Rainbow table attack on a larger database of stolen password hashes (`data/complex-dbdump.json`). Unlike for previous problems, in this step all passwords consist of 4 alphanumeric characters — lowercase only.

Use the following command to run the attack:

```
python attack.py problem3
```

Let n denote the total number of possible passwords, m denote the number of chains in the rainbow table, and k denote the length of each chain.

1. (2 pts) What is the success rate of the attack (percentage of passwords recovered)? Why is it not 100%?

   Your answer here.

2. (3 pts) [Theory] Assume a perfect reduction function family. What is the expected percentage of passwords recovered (assuming passwords are uniformly distributed)? Give an expression in terms of n, m, and k. Justify your answer!

   Your answer here.

3. (5 pts) [Experimental] Keeping $m \cdot k$ constant, run the attack with m = 25000, 50000, 100000, 200000, 400000 and $k = 64, 32, 16, 8, 4$. (You can modify m and k in the `main()` function of `attack.py`.)

   Record the success rate of the attack (p) and write it in the table below. From these results, what do you conclude about the reduction function family?

m	*25000*	*50000*	*100000*	*200000*	*400000*
k	*64*	*32*	*16*	*8*	*4*
p					

   Your answer here.

4. (5 pts) Consider the case $k = 1$. Observe experimentally that the success rate of the attack is lower than $\frac{m \cdot k}{n}$. Why is that? To fix it, change the implementation of `build_rainbow()` to ensure that each chain gets a unique startpoint. (Be careful to ensure that `build_rainbow()` always terminates no matter the values of n, m, and k.)

   Your answer here.

5. (5 pts) [Theory] Assume that a chain can be stored using 8 bytes (4 bytes for the startpoint, 4 bytes for the endpoint). Suppose we set m and k such that $n = m \cdot k$. What is the amount of memory required to store the rainbow table? What is the expected computational complexity of a lookup? Explain your answers!

   Your answer here.

6. (5 pts) [Experimental] Keeping $m \cdot k$ constant, run the attack for m = 25000, 50000, 100000, 200000, 400000 and $k = 64, 32, 16, 8, 4$. Plot the time for a lookup in each case. How does this compare to your answer to the previous question?

   Your answer here.

Problem 4: Bob's custom password hash (30 pts + [bonus] 5 pts)

Your classmate Bob is in charge of the website of the UF chess club. To implement authentication on the website, Bob (who has not taken CNT5410) has decided to roll his own crypto. He has implemented a custom password hash function. Given a password p of even length, p is split into two parts of equal lengths p_1, p_2. The password hash is then computed as:

$$H_l(p_1)||H_l(p_2)||H_l^i(s||c),$$

where s is a random salt, c is a constant string, i and l are positive integers. Here: $H_l(x)$ denotes the first l bytes of the hash of x. This password hash is implemented as `bobs_custom_pw_hash()` in `crypto.py`.

For this problem, you will implement your own attack against Bob's custom password hash scheme. For this you will use `data/bobX-dbdump.json` as database of password hashes.

Use the following command to run your attack:

```
python attack.py problem4
```

1. (5 pts) Is it a good idea to design your own crypto like Bob did in this case? Explain why or why not?

   Your answer here.

2. (5 pts) What are the weaknesses of this custom password hash scheme? Focus specifically on weaknesses that could make this scheme easy to attack.

   Your answer here.

3. (20 pts + [bonus] 5 pts) Implement the best attack you can think of. Place your code in the provided placeholders in `bobs_custom_pwhash_attack()` (`attack.py`). You will be evaluated based on the performance of your attack on passwords of varying length. You can change the size of the targets passwords by changing the value of `pw_length` in the `main()` function of `attack.py`.

   Hint: the fastest attack is neither the naive bruteforce attack nor the one based on rainbow tables.

   Explain how your attack works. How fast is it on passwords of length 4, 6, and 8?

   Your answer here.