CPE-321
Introduction to Computer Security

Assignment: Cryptographic Hash Functions

Objectives

You will explore the properties of cryptographic hash functions and find collisions on a truncated range. The objectives for this lab assignment are as follows:

- To explore Pseudo-randomness and Collision Resistance.
- To break real hashes (To crack a password file).

Programming Tools: Strong recommendation to use Python, PyCryptodome and Bcrypt (or some other high-level language with decent crypto support).

Warning: Be aware that this lab can take considerable time for each task. Based on previous lab experience, the best estimate for task 2 can take 17-18 hours of processing time. Therefore, please do not wait to get this lab started.

Tasks

Please read the following tasks carefully, and finish the lab assignment accordingly:

Task 1: Exploring Pseudo-Randomness and Collision Resistance. In this task, you will investigate the pseudorandom and collision resistant properties of cryptographic hash functions.

a. Write a program that uses SHA256 to hash arbitrary inputs and print the resulting digests to the screen in hexadecimal format.

b. Hash two strings (of any length) whose **Hamming distance** is exactly 1 bit (i.e. differ in only 1 bit). Repeat this a few times.

c. Next we aim to find two strings that creates the same digest (called a collision). Because SHA256 is a secure cryptographic hash function (as far as we know), it is infeasible to use its full 256-bit output. Instead, you will limit its domain to between 8 and 50 bits. **Modify** your program to compute SHA256 hashes of arbitrary inputs, so that it is able to truncate the digests to between 8 and 50 bits (it doesn't matter which bits of the output you choose, as long as you are consistent). Once you have completed the above, try to find a collision in your truncated hash domains (i.e. two different inputs, that create the same, truncated digest). There are at least two different ways of doing this (You need only do one):

    i. Find a target hash collision (weak collision resistance): Give m_0, find $m_1$such that $H(m_0) = H(m_1)$and $m_0 \neq m_1$. This is not hard to code, but it will take time to execute.

    ii. Maximize your chances of finding a collision by relying on the Birthday problem: For any two messages $m_0$and m_1, where $m_0 \neq m_1$, find $H(m_0) = H(m_1)$. This requires a little more code (and memory usage), but will find a collision more quickly. Consider using a hashtable or dictionary, but be careful about efficiency as finding collision on 50-bit outputs is right at the edge of what is feasible by an average computer.

For multiples of 2 bits (i.e. for digests sized 8, 10, 12,...,48, 50 bits), **measure both the number of inputs and total time for a collision to be found**. **Create two graphs: one which plots digest size (along the x-axis)**

to collision time (y-axis), and one which plots digest size to number of inputs. Include these graphs in your report.

Task 2: Breaking Real Hashes. Bcrypt is a hashing algorithm that is based on the blowfish algorithm and is designed to be a slow hash function. You have recovered a shadow file (you can download it from Canvas) which has users stored in the following format:

“User:$Algorithm$Workfactor$SaltHash”

where salt is 22 characters base64 encoded and hash is the remainder.

For example:

“Bilbo:$2b$08$L.z8uq99JkFAvX/Q1jGRI.TzrHIIxWMoRi/VzO1sj/UvVFPgW8dW.”

represents:

User: Bilbo

Algorithm: 2b or bcrypt

Workfactor: 8

Salt: L.z8uq99JkFAvX/Q1jGRI.

Hash value:TzrHIIxWMoRi/VzO1sj/UvVFPgW8dW.

This file is generated using the bcrypt library. Each user chooses a password that is a single word from the nltk word corpus between 6 and 10 letters long. **Your job** is to crack each user’s password. You can’t use any password cracking tools. Your solution should be a custom cracking script. Record how much time it takes to crack the password for each user.

Implementation hints: bcrypt is meant to run slow. There is no amount of optimized code you can implement to make bcrypt fast, as this would defeat the point of bcrypt slowing down the attacker (you!) in cracking passwords. It’s completely fine to patiently wait many hours for your program to crack passwords. However, if you want it to run faster, your only choices are to run it on a faster CPU or run it on multiple CPU cores, since the dictionary lookup is highly parallelizable.

Here’s some approximate numbers to help you estimate how long your hashing program will run. On a MacBook Air with M1 chip, each hash at workfactor 8 takes 30 ms, workfactor 9 takes 60 ms, workfactor 10 takes 110 ms, workfactor 11 takes 220 ms, workfactor 12 takes 420 ms, workfactor 13 takes 840 ms. The word corpus dictionary of 6-10 letter words is about 135,000 words. You can do the math.

If you want to run hashing in parallel, split up the dictionary into chunks and in each core and/or thread, match the user’s hash against the hashes of the words in the dictionary chunk, and see which core/thread finds the matching hash first. If you don’t want to implement multiprocessing, you could hash the dictionary chunks on different physical machines you have hanging around. If you run your hashing on 10 cores/threads/machines you will get your answer 10 times faster. Do not bother to parallelize different user password hashes across different cores, you will not see the same speedup, as some user hashes take many hours and some only take a few minutes.

Questions

1. What do you observe based on Task 1b? How many bytes are different between the two digests?
2. What is the maximum number of files you would ever need to hash to find a collision on an n-bit digest? Given the birthday bound, what is the expected number of hashes before a collision on an n-bit digest? Is this what you

observed? Based on the data you have collected, speculate on how long it might take to find a collision on the full 256-bit digest.

3. Given an 8-bit digest, would you be able to break the one-way property (i.e. can you find any pre-image)? Do you think this would be easier or harder than finding a collision? Why or why not?
4. For Task 2, 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? What about word1:word2:word3? What about word1:word2:number where number is between 1 and 5 digits? Make sure to sufficiently justify your answers.

In Your Report

Please address the following in your report:

1. What you did and what you observed;
2. Include all code that you wrote;
3. Please include answers to all questions;
4. Please include any explanations of the surprising or interesting observations you made;
5. Write at a level that demonstrates your technical understanding, and do not shorthand ideas under the assumption that the reader already "knows what you mean". Think of writing as if the audience was a smart colleague who may not have taken this class;
6. Describe what you did in sufficient detail that it can be reproduced. Please do not use a screenshot of the VM to show the commands and code you used, but rather paste (or carefully retype) these into your report from your terminal. Submit your completed write up to Canvas in PDF format. Also remember

that a picture can be worth a thousand words to graphing your results with good captions can be beneficial.