CS 453/698 Assignment 4: Authentication and Hashing

TA: Cong Ma <cong.ma@uwaterloo.ca>
Office Hours: See Piazza announcement

Release Date: July 15, 2025
Due Date: July 25, 2025

Overview

In this assignment, you will explore how hash functions are used in authentication systems through hands-on attacks. Each of the 4 problems provides you with the source code of a login program that uses a specific hashing scheme; each problem also contains a `password.txt` file that stores a single password hash.

Your goal is to implement an `inverse` program that recovers a password from a hash. While the assignment itself is implemented in Python, you may write your solution in any language of your choice.

You might want to review the slide: `https://watssec.github.io/cs453-s25/assets/modules/user-sec/auth/slides.pdf`

Starting Code

The starting code is hosted on CS 453/CS 698 LEARN → Content → A4 Files

Commands

- `./login PASSWORD`
  It reads the hash from `password.txt`, and reads a password from the command line. It uses a specific hash function to hash the given password and compare with the one on file. It will print either `Login successful!` or `Login failed!`.
- `./invert HASH`
  Your solution goes in here. The program reads a hash from the command line and prints a password that can be used in `./login`; it effectively inverse the hash function used in `./login`.

If you use an interpreted language, make sure `./invert` contains the correct shebang. If you use a compiled language, `./invert` should be the compiled binary.

- `./test`
  Runs an automated test on your invert implementation. It only runs a single test case; if you want more test cases, you can manually generate hashes using the hash function inside `./login` and use them to replace the content of `password.txt`, then run `./test` again.

Other Files

- `password.txt`: Contains the hash used by `./login`. **A different hash will be used during grading.**
- `rainbow_table.txt` (Q3 only): Maps unsalted scrypt hashes to their corresponding passwords.
- `pw_dict.txt` (Q4 only): A dictionary of possible passwords.

Problems

For Q1-Q4, write a short reflection in the `./writeup.txt` file.

Q0 (0 points):

This is a warm-up where the password is stored in plaintext. A sample invert implementation is provided, which simply returns its input. Try running `./login`, `./invert`, and `./test` to familiarize yourself with the assignment setup.

Q1 (5 point):

This `./login` uses a *bad* hash function. Implement `invert` to find a password that matches the hash in `password.txt`.

Reflect on why this hash function is insecure; which property does this hash function lack?

Q2 (5 point):

Similar to Q1, but with a different *bad* hash function.

Reflect on why this hash function is insecure; which property does this hash function lack?

Q3 (5 point):

This login uses a secure cryptographic function (scrypt) but without a salt. A precomputed rainbow table is provided in `rainbow_table.txt`, and the password is guaranteed to be in it. Use it to implement `invert`.

Reflect on how a user can be protected from rainbow table attacks.

Q4 (5 point):

This problem uses a salted hash. A password dictionary is provided in `pw_dict.txt`, and the correct password is among them. Implement `invert` by brute-forcing with this dictionary.

Reflect on how the use of salt improves security; why is cracking hash harder in Q4 than in Q3.

Deliverable

Package the entire assignment directory using `zip -r cs453_a4.zip cs453_a4` and submit it to the LEARN Dropbox.

Ensure the following files are included:

- Executable `invert` programs:
  - `q1/invert`, `q2/invert`, `q3/invert`, `q4/invert`
- (Only if you use a compiled language) Source code of the `invert` programs; briefly describe how to compile your program in `./writeup.txt`. e.g.:
  - `q1/invert.c`, `q2/invert.c`, `q3/invert.c`, `q4/invert.c`
- Write-up:
  - `./writeup.txt`