CS 551 Systems Programming Project 1

February 10, 2025

1 Introduction

CAREFULLY READ THE *ENTIRETY* OF THIS DOCUMENT. DEVIATIONS FROM THE INSTRUCTIONS IN THIS DOCUMENT *WILL* BE PENALIZED.

IF YOU HAVE QUESTIONS OR NEED CLARIFICATION ON SOMETHING IN THIS PROJECT DESCRIPTION, YOU *MUST* INCLUDE THE RELEVANT TEXT FROM THIS PROJECT DESCRIPTION YOU ARE ASKING ABOUT IN YOUR EMAIL, OR WE WILL ASSUME THAT YOU HAVEN'T FULLY READ THE PROJECT DESCRIPTION AND *WILL NOT* RESPOND. LACK OF RESPONSE TO EMAILS THAT DO NOT FOLLOW THIS REQUIREMENT WILL NOT BE CONSIDERED AN EXCUSE FOR DEVIATING FROM THE DIRECTIONS OUTLINED IN THIS DOCUMENT.

2 Hashassin

We will be working on a library (and front end) called *Hashassin*. Hashassin is a tool for generating and cracking password hashes. For Project 2, you will be adding functionality to crack passwords using rainbow tables, and for Project 3 you'll add networking, but for Project 1 we are going to be working on the basic building blocks.

This is a group project with a maximum of *two* students per group.

NB: Group size is non-negotiable. Attempts to negotiate group size will be considered attempting to acquire an unearned grade.

3 Deliverables

There is one deliverable for this project.

1. Project 1 code
    - GitHub Classroom Section: `https://classroom.github.com/a/7ZFsTk8-`
    - March 18th, 11:59PM. **NB: This is a hard cut off and you will not be able to commit to this repository when the due date passes without a late penalty. You should add at least your `HONESTY.md` file to this repository as soon as you accept it to ensure you can do a pull request if you are late.**

4 Instructions

This project is relatively straight forward as we are mostly interested in making sure that you are able to write a relatively complicated program that uses threading and deals with IO. To that end, you will be creating a workspace with two crates, one binary and one a library.

Your binary crate should be in a directory called `cli` and it should compile to a binary named `hashassin`.

Your library crate should be in a directory called `core`, available in a namespace called `hashassin-core`, and usable by other crates via `use hashassin_core::Whatever`.

Your CLI will support three commands: 1) `gen-passwords`, 2) `gen-hashes`, and 3) `dump-hashes`.

`gen-passwords` will generate random passwords and save them to a file. `gen-passwords` *must* have several options that can be set:

1. `--chars`, which specifies the number of characters to be used in generated passwords. This value *must* be greater than zero and should be 8 bits wide. `--chars` should have a default value of 4.
2. `--out-file`, which, *if present*, will write the output of the command to the specified file, one passwords per line (using Unix new lines and no empty lines between them). If the specified output file already exists, it should be overwritten/truncated. If not present, results should be written to `stdout`.
3. `--threads`, the number of threads to use to generate passwords. This value *must* be greater than zero and the maximum value should be the maximum length of an array on whatever system the program is being run on. `--threads` must have a default value of 1.
4. `--num`, which is the number of passwords to generate. It must be greater than zero and the maximum value should be the maximum size of an array on the system it's being run.

All passwords generated should be valid ASCII, both caps and lower case, including punctuation and spaces, but *not* including non-printable characters like `\n` or `\a`.

NB: You can add additional functionality that can restrict or expand the set of characters to be used in generated passwords, but by default, generated passwords should adhere to the above guidelines.

`gen-hashes` will generate hashes from a set of input passwords. `gen-hashes` *must* have several options that can be set:

1. `--in-file`, which specifies the path to read plaintext passwords from, one password per line. The length of the first password in the file should be assumed to be the length of all the passwords in the file. If a non-conforming password exists further in the file, your program should error out (not panic).
2. `--out-file`, which will write the output of the command to the specified file according to the output file format specified below. If the specified output file already exists, it should be overwritten/truncated.
3. `--threads`, the number of threads to use to generate hashes. This value *must* be greater than zero and the maximum value should be the maximum length of an array on whatever system the program is being run on. `--threads` must have a default value of 1 and be a non-zero number between 1 and the maximum size of an array on the system it's being run (inclusive).
4. `--algorithm`, which should be the algorithm used to generate hashes.

Output file format: Your output file *must* conform to the following format. Non-conformant output will prevent you from earning points.

The file format format is simple and consists of a few fields.

- `VERSION`: The first byte of your output file *must* be a version number. Unless otherwise specified in a future update, this should always be `1`.
- `ALGORITHM LENGTH`: The second byte of your output file should be the length of the the ASCII encoded string which starts at the 3rd byte (see next field).
- `ALGORITHM`: Starting at the third byte, your output file should include an ASCII string representation of the hashing algorithm that was used to generate the file, in all lowercase. **NB:** this is *not* a null terminated string!
- `PASSWORD LENGTH`: The byte following the name of the hashing algorithm must be the number of characters in the passwords this list of hash was generated from.
- `DATA`: The remaining bytes in the file should be the hashed passwords with zero padding between them.

`dump-hashes` will take as input a generated hashes file and dump it to plaintext. `dump-hashes` has one and only one parameter option:

1. `--in-file`, which takes a path to the file generated from `gen-hashes` that will be dumped to stdout.

Output file format: Your output file *must* conform to the following format:

Line 1: "VERSION: $VERSION_NUMBER", where $VERSION_NUMBER is the version number in the supplied input file.

Line 2: "ALGORITHM: $ALGORITHM", where $ALGORITHM is the name of the algorithm as specified in the input file.

Line 3: "PASSWWORD LENGTH: $PASSWORD_LENGTH", where $PASSWORD_LENGTH is the password length specified in the input file.

Remaining lines should be a hex (in the case of `md5`, `sha256`, and `sha3_512`) or string (in the case of `scrypt`) encoded hash of each of the hashes in the input file, one hash per line.

For example, the following output for an input file:

```
VERSION 1
ALGORITHM: md5
PASSWORD LENGTH: 4
18c07a5177752088fe532ccb79a19963
```

You are free to add other options and functionality, but whatever you add should not be required to be set by a user. I.e., it should either be optional or have sensible defaults.

5 Grading

This project is relatively straight forward. You will receive points according to the following list. Please note that there are more than 100 points available.

-1,000 points: If you do not update `CREDITS.md` with the names of your group members, as well as an honest break down of the work each group member did, you will receive **NEGATIVE ONE THOUSAND** points. I.e., it would be next to impossible to get anything higher than a zero on this project.

-1,000 points: If you do not include an `HONESTY.md` with the academic honesty statement from the syllabus, you will receive **NEGATIVE ONE THOUSAND** points.

-1,000 points: If you do not include a `README.md` with information about your project, libraries used, instructions on how run things, etc., you will get **NEGATIVE ONE THOUSAND** points. Your `README.md` should also include the items in the grading rubric that you attempted. Only items you note in your `README.md` file will be graded!

-1,000 points: If running `cargo fmt` results in any change to your repository, you will get **NEGATIVE ONE THOUSAND** points. Be sure to run `cargo fmt`!!!!!!!!!

-1,000 points: Program does not compile to correct binary/library name.

25 points: Program compiles.

20 points: **All** required functionality of `gen-passwords` is implemented (e.g., setting `--threads` actually uses multiple threads, etc.)

- 5 points: `--chars`
- 5 points: `--out-file`
- 5 points: `--threads`
- 5 points: `--num`

25 points: **All** required functionality of `gen-hashes` is implemented (e.g., setting `--threads` actually uses multiple threads, etc.). **NB:** This includes output in correct format.

- 5 points: `--in-file`
- 5 points: `--out-file`
- 5 points: `--threads`
- 5 points: `--algorithm`
- 5 points: Support the `md5`, `sha256`, `sha3_512`, and `scrypt` hashing algorithms.

10 points: **All** functionality required of `dump-hashes` is implemented.

2.5 points: Comprehensive documentation. Points will be determined by looking at the documentation built when running `cargo doc --document-private-items --no-deps`.

5 points: No warnings from `cargo check` with no use of directives that would suppress default warnings (e.g., `#[allow(dead_code)]`)

5 points: No warnings from `cargo clippy` with no use of directives that would suppress default warnings (e.g., `#[allow(dead_code)]`)

10 points: Support hashing algorithms in a *generic* fashion. It is difficult to make things completely generic, but partial credit is available here.

0.25 points: Support an additional hashing algorithm (0.25 points per additional algorithm).

2.5 points: Proper error handling. Create and use proper error types in library code and appropriate handling in any user code (e.g., CLI).

5 points: No unwraps or expects. Add `#![deny(clippy::unwrap_used, clippy::expect_used)]` to the top of your files. If running `cargo clippy` doesn't result in any errors, then you get the 5 points.

2.5 points: Proper logging. We suggest using the `tracing` crate to generate logs. Points will be determined on comprehensiveness of logs, as well as the usage of different log levels.

Unlimited points: Cool factor. Points are determined subjectively by me, but if I find anything about your project to be particularly unique, difficult, clever, etc., you can get some extra points. Be sure to point out anything we should pay special attention to in your `README.md` file.

5.1 Allowed Crates

You are allowed to use the `hex` crate (`https://crates.io/crates/hex`).

Generally speaking, you are free to use any crates listed on `https://blessed.rs`.

Any other crates not on Blessed.rs need to be approved by Jeremy.

Be sure to note what crates you use in your `README.md`!