CITS3007 Project 2022

Version:	0.1
Date:	28 Sep, 2022

Introduction

- This project contributes **30%** towards your final mark this semester, and is to be completed as individual work.
- The project is marked out of 140.
- The deadline for this assignment is listed on the assessments page of the CITS3007 at https://cits3007.github.io/assessment/, and is 5:00 pm Thursay 20 October.
- You are expected to have read and understood the University Guidelines on Academic Conduct. In accordance with this policy, you may discuss with other students the general principles required to understand this project, but the work you submit must be the result of your own effort.
- You must submit your project before the submission deadline above. There are significant Penalties for Late Submission (click the link for details).

Items to submit

You will need to submit:

1. a PDF report, `report.pdf`
2. a text file containing the answer to a question (`answers.txt`), and
3. a C file, `adjust_score.c`

Do not submit a .zip or .tar file or other archive.

PDF report Your PDF report should use A4 size pages. The font for body text should be between 9 and 12 points. The report should contain numbered headings, with useful heading titles. Each question should be answered on a new page. Any diagrams, charts or tables used must be legible and large enough to read when viewed on-screen at 100% magnification. All pages (except the cover, if you have one) should be numbered. If you give scholarly or other references, you may use any standard citation style you wish, as long as it is consistent. Cover sheets, diagrams, charts, tables, bibliographies and reference lists do not count towards any page-count maximums.

text file The format for the text file is given in Part 1.

C files Your code is expected to be clearly written, well-formatted, and easy for others to understand. It should follow sound programming practices, including:

- the use of meaningful comments
- well chosen identifier names
- appropriate choice of basic data-structures, data-types, and functions
- appropriate choice of control-flow constructs
- proper error-checking of any library functions called, and
- cleaning up/closing any files or resources used.

It should compile without errors or warnings with `gcc`, using the compilation flags "`-std=c11 -pedantic -Wall -Wextra`".

Documenting assumptions If you believe there is insufficient information for you to complete part of this project, you should document any *reasonable assumptions* you had to make in order to answer a question or write a method.

In that case, make sure your report has as one of its first sections a section entitled "Assumptions", in which you list these, giving each one a number and explaining why you think it's a reasonable assumption.

Then in your code or later in your report, you can briefly refer to these assumptions (e.g. "This test case assumes that Assumption 1 holds, so that we can ... ").

Part 1: CVE-2019-17498 (70 marks)

For this part of the project, you will need to investigate CVE-2019-17498. In your report, you should include a section on this CVE, addressing the following questions.

a. What product or software package does CVE-2019-17498 affect? What type of vulnerability or vulnerabilities does the CVE represent? (10 marks)
b. How severe is the impact of the CVE? What factors go into assessing this impact? (10 marks)
c. Describe how the CVE could be exploited, and what the consequences of a successful exploit could be. (20 marks)
d. Show the C source code that gives rise to the vulnerability described by this CVE. On the first line of your "`answers.txt`" file, provide a URL for the vulnerable version of the C file, then a space, then a range of line numbers (two numbers separated by a dash). (You may also include these in your report if you wish.) (10 marks)
e. Explain what changes were made in the affected product to address the vulnerability described by the CVE, and how they work. (20 marks)

For each question you should *cite your sources*. (These may be traditionally published works, but are more likely to be webpages.)

(Maximum length: 5 pages)

Part 2: setuid program (70 marks)

You are employed by a games publisher which develops the popular game *Curdle*, in which players are provided with images of various fermented or coagulated milk–based products – cheeses and yoghurts – and are required to guess the particular product.

Although available for multiple platforms, the most popular one on which Curdle is available is Linux (specifically, Ubuntu 20.04, running on the x86-64 platform). On Linux, the files used by the game (for instance, a file storing scores achieved by different players) are owned by the user `curdle`.

Your task is to write a small portion of the game executable which modifies a *scores* file used by the game. Running `ls -l` on the scores file produces the following listing:

```
-rw------- 1 curdle curdle 2602 Sep 21  2021 /var/lib/curdle/scores
```

Every line in the `scores` file is 20 characters long: 10 characters for a player name of maximum length 9, ending in a `NUL`, and 10 characters for a score. (For player names shorter than 9 characters, you may assume the name "field" is padded with `NUL` characters to ensure it's 10 characters in length total.)

Write a file `adjust_score.c` containing a single function, `adjust_score()`, with the following signature

```
int adjust_score(uid_t uid, const char * player_name,
                 int score_to_add, char **message)
```

- `uid` contains the user ID of the `curdle` user
- `player_name` is the name of the current player
- `score_to_add` is the player's score for this game
- the caller should pass in the address of a pointer-to-char to `message`, in which `adjust_score` can write an error message (if needed).

You should assume the program will be run as a setuid program owned by the user `curdle`, and that the first thing the `main` function for the game does is to call `seteuid()` to set the effective UID to the real UID (i.e., to drop privileges), and then call `setegid()` to do the same for the group ID.

The `adjust_score` function shoud do the following:

- Open the scores file (using appropriate privileges)
- Read through the scores file for a line starting with `player_name`.
- If such a line is found, your program should read the `int`, derive a new score by adding the value in `score_to_add`, then replace the original line with a new line containing the new score. (Hint: you will want to use `fseek()`.)
- If such a line is not found, add a new, valid score line to the end of the file containing the player name and score (which is simply `score_to_add`).
- The function should then drop any privileges used.

If the score was changed successfully, the function should return 1; if not, it should return 0, and:

- allocate memory for an error message using `malloc()`
- write an error message to that memory, and
- set `*message` to the newly allocated memory.

If the new score would overflow an `int`, that counts as an error – the situation should be checked for and an appropriate error message printed.

You may use helper functions if desired which are called by the `adjust_score` function.

Your code will be awarded 50 marks for correct operation, and 20 marks for code concision and clarity. It should avoid introducing any security vulnerabilities.